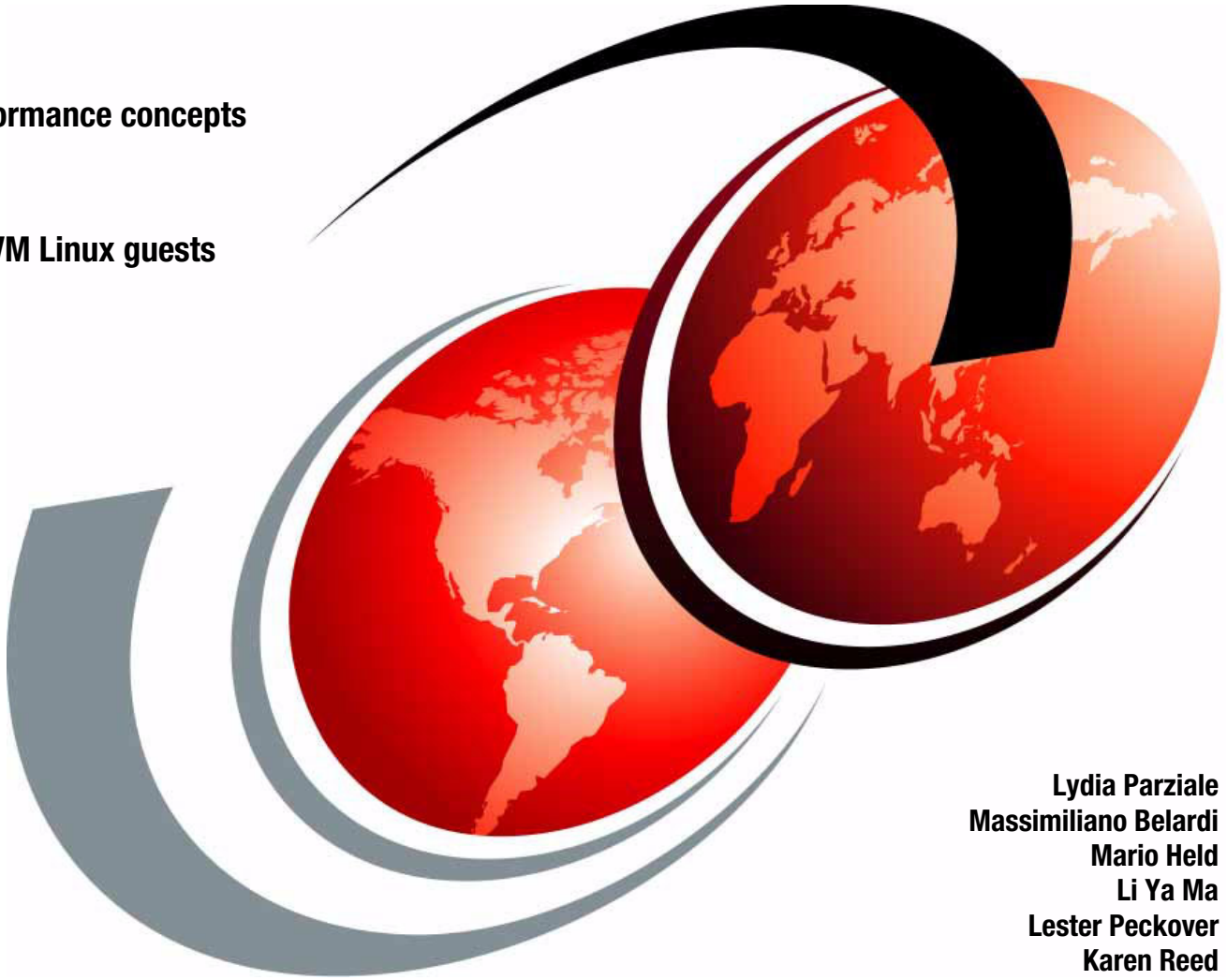
IBM

Linux on IBM System z: Performance Measurement and Tuning

- Understanding Linux performance on System z
- z/VM performance concepts
- Tuning z/VM Linux guests



Lydia Parziale
Massimiliano Belardi
Mario Held
Li Ya Ma
Lester Peckover
Karen Reed

Redbooks

ibm.com/redbooks

International Technical Support Organization

Linux on IBM System z: Performance Measurement and Tuning

February 2008

SG24-6926-01

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Second Edition (February 2008)

This edition applies to Version 5, Release 3 of z/VM (product number 5741-A05) and Linux SLES10.

© Copyright International Business Machines Corporation 2003, 2008. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2003, 2008. All rights reserved.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2003, 2008. All rights reserved.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo)  ®
developerWorks®
eServer™
z/OS®
z/VM®
zSeries®
z9™
DirMaint™
DB2®
DS6000™
DS8000™
Enterprise Storage Server®
ECKD™
ESCON®
FICON®
GDDM®
HiperSockets™
Hypervisor™
IBM®
OMEGAMON®
OS/390®
PR/SM™
Redbooks®
S/390®
System p™
System x™
System z™
System z9®
Tivoli®
TotalStorage®
VM/ESA®
WebSphere®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Java, JavaServer, JavaServer Pages, J2EE, PDB, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication discusses performance measurement and tuning for Linux® for System z™. It is intended to help system administrators responsible for deploying Linux for System z understand the factors that influence system performance when running Linux as a z/VM® guest.

This book starts by reviewing some of the basics involved in a well-running Linux for System z system. An overview of some of the monitoring tools that are available and some that were used throughout this book is also provided.

Additionally, performance measurement and tuning at both the z/VM and the Linux level is considered. Some tuning recommendations are offered in this book as well. Measurements are provided to help illustrate what effect tuning controls have on overall system performance.

The system used in the writing of this book is IBM System z9™ running z/VM Version 5.3 in an LPAR. The Linux distribution used is SUSE Linux Enterprise Server 10. The examples in this book use the Linux kernel as shipped by the distributor.

The z9 is configured for:

Main storage	4 GB
Expanded storage	1 GB
Minidisk cache (MDC)	250 MB
Total LPARs	45
Processors	Four shared central processors (CPs) defined as an uncapped logical partition (LPAR)

The direct access storage devices (DASD) used in producing this IBM Redbooks publication are 2105 Enterprise Storage Server® (Shark) storage units.

The intent of this book is to provide guidance on measuring and optimizing performance using an existing zSeries® configuration. The examples are intended to demonstrate how to make effective use of your zSeries investment. The workloads used are chosen to exercise a specific subsystem. Any measurements provided should not be construed as benchmarks.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Lydia Parziale is a Project Leader for the ITSO team in Poughkeepsie, New York, with domestic and international experience in technology management including software development, project leadership, and strategic planning. Her areas of expertise include e-business development and database management technologies. Lydia is a Certified IT Specialist with an MBA in Technology Management and has been employed by IBM for 24 years in various technology areas.

© Copyright IBM Corp. 2003, 2008. All rights reserved.

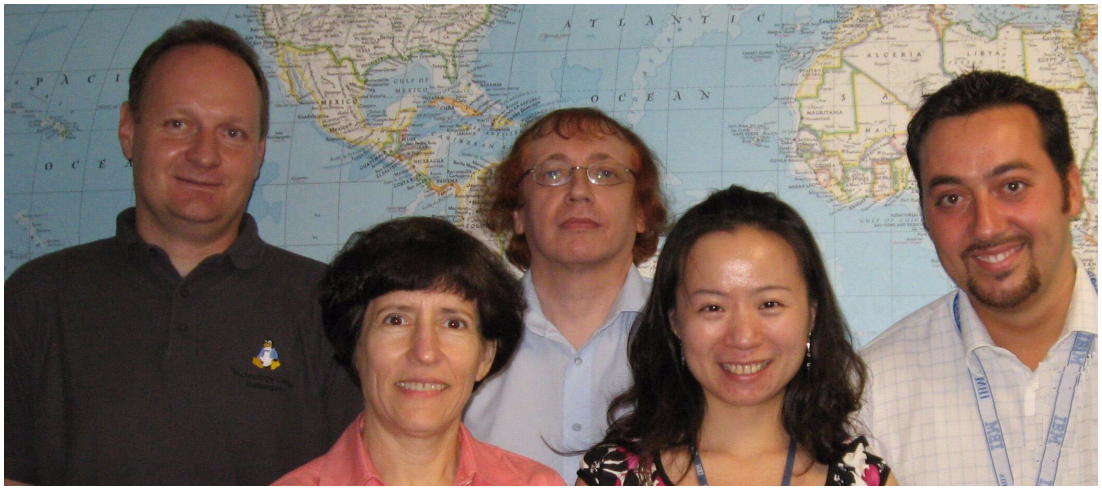


Figure 1 The team that wrote this book: (from left) Mario Held, Karen Reed, Lester Peckover, Li Ya Ma (Maria), Massimiliano Belardi (Max)

Massimiliano Belardi is an Advisory IT Architect supporting IBM System & Technology Group in Italy. He joined IBM in 1998 and his areas of expertise include design and presales support for System z and Linux-based architecture. He is strongly committed to supporting the growth of the System z platform and Linux Solutions adoption.

Mario Held is a Performance Analyst for the Linux on System z - System & Performance Evaluation Development in the IBM development lab in Boeblingen, Germany, since 2000. His area of expertise is the general performance of Linux on System z, and he specializes in gcc and glibc performance. He presents worldwide on all areas of performance. Between 1997 and 2000 he was a System Developer. For six years before joining the IBM development lab he was an application programer on the mainframe with a health insurance company in Germany.

Li Ya Ma has been a IT Specialist for the IBM ATS team of China since 2005. She has domestic and international experience in performance benchmark on both Linux on System z and WebSphere® Application Server on z/OS®. She gives workshops and presentations, writes articles on the Linux on System z, from the basics to performance tuning. She is committed to supporting new growth of the System z platform and Linux solutions adoption in China.

Lester Peckover is an Infrastructure, Enterprise Performance, and Capacity Architect in the United Kingdom. He has 32 years of experience in the IT field, during that time covering IBM System x™ and System p™, mainly specializing in System z; and, since 2000, Linux running on the mainframe. He has also worked on a number of other system platforms in performance and many other areas, specializing in virtualisation. He been with IBM for 18 years, and is a Senior IT Specialist. He started his academic training with a bachelor's honors degree in Biochemistry, and then switched to research-based computational chemistry for both his masters and his doctorate. His areas of expertise include performance analysis and modelling. He helped shape the tool now known as IBM VM Performance Tool Kit in its early, first decade of development, as well as other IBM VM program products and tools from other vendors. He has written and also taught and presented extensively on performance and capacity, plus various other systems management topics, in many countries across the world.

Karen Reed is an IT Specialist supporting IBM Tivoli® software in San Francisco, California. She has over ten years of experience in systems performance monitoring and automation software for both System z and distributed systems**.**

Thanks to the following people for their contributions to this project:

Roy P. Costa
International Technical Support Organization, Poughkeepsie Center

Ursula Braun
Linux for zSeries Development, IBM Germany

Patricia Rando
System z - Core Technologies, IBM US

Giorgio Acquafresca
zSeries Technical Support, IBM Italy

Klaus Bergmann, Juergen Doelle, Horst Hartmann, Mustafa Mesanovic
Linux on System z - System & Performance Evaluation Development, IBM Germany

Carsten Otte
Linux file systems and virtualization development, IBM Germany

Special thanks to our technical review team:

Bruce Hayden, Alan Altmark, Sam Cohen, Jason Herne, Stephen Wehr, Jan Glauber, Martin Kammerer, Horst Hummel, Stefan Weinhuber, Eberhard Pasch

Additional thanks to the authors of the previous edition of this book. Authors of the first edition, *Linux on IBM System z: Performance Measurement and Tuning*, published in May 2003, were Gregory Geiselhart, Laurent Dupin, Deon George, Rob van der Heij, John Langer, Graham Norris, Don Robbins, Barton Robinson, Gregory Sansoni, and Steffen Thoss.

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-6926-01
for Linux on IBM System z: Performance Measurement and Tuning
as created or updated on February 28, 2008.

February 2008, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- Moved all of the Linux monitoring tools to their own chapter
  - Added Omegamon XE for z/VM and Linux
  - Added performance analysis tools such as SAR, vmstat, and iostat
- Moved tuning to its own chapter
  - Added bottleneck analysis
  - Added sizing considerations
  - Added benchmark concepts and practices
- Added information about emergency scanning
- Added information about new disk and network features

Changed information

- Updated information specific to z/VM to the current releases of v5.2 and v5.3
- Updated information specific to Linux kernel 2.6

© Copyright IBM Corp. 2003, 2008. All rights reserved.

1

Virtualization and server consolidation

In this chapter, we discuss what virtualization is and what benefits we can get by running Linux as a z/VM guest. We examine sharing real hardware resources using the virtualization technology provided by z/VM.

© Copyright IBM Corp. 2003, 2008. All rights reserved.

1.1 Server consolidation and virtualization

The concept of virtualization, for example, where large numbers of virtual servers are consolidated onto a small number of relatively large, physical servers, is rapidly becoming commonplace in data centers around the world. The System z mainframe is ideally suited for this large hosting server role.

Selection of workloads suitable for virtualization needs to be done at a very early stage when planning to move to this shared pool resource environment, as opposed to running each workload on many discrete servers with non-shared resources. Some workloads involve mainly number crunching activity, for example, financial forecasting applications. Any application that relies mainly on using a single resource particularly heavily, in this case CPU with relatively low I/O and memory requirements, is not a good candidate for virtualization. The more commonly encountered workloads that use CPU, I/O, and memory in a more moderate, balanced fashion over time are likely to be good candidates, as they can coexist far more readily in a shared resource infrastructure.

We need to be wary of misleading comparisons, especially those that only consider one of the major resources. If we compare the raw cycle speed of a System z CPU with an average Pentium® processor or a single processor on a multi-core processor chip, the comparison might appear to suggest that System z servers would not have a great advantage. If that workload uses the full 100% of a modern PC for 24 hours a day, the same workload could easily use most of an entire System z CPU for the whole day. Virtualizing this workload means effectively dedicating part of the CPU resource to a single workload, which is clearly not conducive to coexistence in a more ideally virtualized, shared pool environment.

Even though a System z machine has multiple CPUs, with a higher maximum number than any other platform, that could actually allow the application to run very well, it is still not advisable to host such a workload on System z hardware. Fortunately, in terms of real-world processing, such extremes are not typical, and so there are very many other situations where using System z *does* make a lot of sense. In this book, we use a case study example that utilizes an IBM WebSphere workload generated by the Trade 6 Benchmark to represent a typical balanced workload that is well suited to virtualization.

Consider the case where the workload does not use the machine for the full 24 hours, but only for 12 hours. Imagine that there is another similar workload using the same amount of resources, but during the other 12 hours of the day. In that situation, we can put both Linux systems on the same System z and let them use all the CPU cycles that they need. These two systems would happily coexist on the same hardware, running as virtual server guests. As we see in the examples presented later in this book, they are using in total around half of the purchased CPU power than would be the case if they are running on freestanding, discrete hardware.

Total cost of ownership (TCO) is also another key factor when making the decision to base a virtualized environment on the mainframe. When all factors are taken into account, including hardware cost, software licensing, on-going support costs, space usage, and power costs related to running and cooling the machine room area, the System z option is likely to deliver the overall lowest TCO for many installations with a currently large investment, comprising many distributed servers. In addition, the reliability, availability, and serviceability that are known and proven strengths of the mainframe are additional positive factors.

When multiple Linux systems run on the same System z, each Linux system is made to believe that it has dedicated access to a defined portion of the System z machine, using a technique known as timesharing. The System z hardware and z/VM take care of the smoke and mirrors without necessarily dedicating portions to that Linux system in practice.

Each Linux system runs in its own *virtual machine*. The characteristics of that virtual machine (memory size, number of CPUs) define the hardware that Linux sees. The allocation and tuning controls in z/VM specify how real hardware resources are allocated to the virtual machine.

1.1.1 Virtualization of the CPU

CPU virtualization is accomplished by timesharing. What this means is that each Linux guest, in turn, gains access to a CPU for a period of time. After that time slice is completed, the real CPU is free to run the work of the other Linux system. This cycle continues over time, with each system being serviced as it generates more and more work.

The cost of running both of these workloads on discrete servers is twice the cost for running only one of them, but with System z, by sharing the resources effectively, we run both workloads for the price of one. Real-world business applications in practice often run far less than 50% of the time, so that allows you to run even more of these workloads on the same System z machine. The workload from these business applications is most likely also not in a single, sustained burst per day, but in multiple short ones over the entire day. Given a large enough number of servers and short enough intervals of workload, the chances of spreading the total CPU requirements over the full day become better.

Because the System z is specifically designed to do timesharing, z/VM can switch between many system tasks in a very cost-effective manner. The evolution of this role began many decades ago, and the z/VM we see today contains all of the refinement and high levels of reliability expected in the modern world of information technology.

1.1.2 Virtualization of memory

In addition to the CPU requirements discussed, the workload also has memory requirements. The amount of memory needed to contain the Linux system and allow it to run is the working set size. Virtualization of memory on z/VM is done by using a technique called paging. By placing the memory in a central shared pool, the Linux systems take turns using the main storage (also known as main memory) of the machine. Paging volumes residing on disks, and expanded storage is used to hold the pages of the inactive Linux systems so that they can be brought into main storage by z/VM when needed.

The challenge for z/VM is to be able to bring in the working set of a Linux system in the shortest possible time when a Linux system has work to do, and fortunately it has the best architecture available, in terms of central and expanded storage, and advanced I/O subsystem, to be able to deliver the required results.

1.1.3 Virtualization of the network

When applications and servers want to connect to network servers or to other servers, the necessary resources must be planned and put into place. All the network resources, such as IP addresses, network adapters, LANs, and bandwidth management, can be virtualized in z/VM. With virtualization, network elements can be pooled and shared to make communication across the IT infrastructure faster, more efficient, cost-effective, secure, flexible, and available. Outages due to physical network device failure or software failure in the device are eliminated.

Virtual LAN (VLAN) is used to provide a network infrastructure inside z/VM. VLAN provides the isolation required to ensure that data would not be interspersed when flowing across the shared network. Guests in the same VLAN communicate with each other directly, without the need for any routing outside of the box, or any intermediate hardware, which would otherwise

cause much latency. In fact, guests in the same VLAN can talk to each other with cross-memory speed.

1.1.4 Levels of virtualization

The virtual machine provided by z/VM to run the Linux system is not the only virtualization that is taking place. The Linux system itself runs multiple processes, and the operating system allocates resources to each of these processes to allow them to get their work done. Some of the processes running on Linux run multiple tasks and allocate their resources to those tasks. If the z/VM system runs in a logical partition (LPAR), which again uses the timesharing principle, z/VM also uses only part of the real hardware.

These multiple layers of virtualization can make it hard for an operating system to find the best way to use the allocated resources. In many cases, tuning controls are available to solve such problems, and allow the entire system to run very efficiently. When the operating system was not originally designed to run in a shared environment, some of the controls turned out to have surprisingly negative side effects.

1.1.5 Benefits of virtualization

As IT systems are getting larger and more complicated, the cost of managing these systems is growing faster than the cost of new hardware for them. The primary concern of management is to decrease the costs and increase the revenue in the meanwhile. Virtualization can contribute to the overall management by lowering the cost, compared to the existing infrastructure, and also by reducing its complexity and improving its flexibility to quickly respond to the business needs.

More and more users realize that they can get the following benefits from virtualization by consolidating hundreds of existing low-utilized and unstable servers from their production and development environment on to one or a small number of System z boxes:

- Higher resource utilization

  Running as guests on z/VM, several Linux servers can share the physical resources of underlying box, resulting in higher levels of sustained resource utilization, comfortable even when approaching 100% for CPU. This is especially relevant for variable workloads whose average needs are much less than an entire dedicated resources and do not peak at the same time. Two workloads, which are only running in different halves of the day, are a very good example.

- More flexibility

  Virtualization can provide Linux guests with great flexibility. Linux servers can dynamically reconfigure resources without having to stop all of the running applications.

- Improved security and guest isolation

  Each virtual machine on z/VM can be completely isolated from the control program (CP) and the other virtual machines. If one virtual machine crashes, none of the others is affected. Data is prevented from leaking across virtual machines and applications can communicate only over configured network connections.

- Higher availability and scalability

  Linux servers running on z/VM can improve their availability due to the reliability of underlying mainframes, which have refined hardware and software architectures that have evolved over four decades. Physical resources can be removed, upgraded, or changed without affecting their users. z/VM enables its Linux guests to be easily scaled-up or scaled-out to accommodate ever-changing workload demands.

- Lower management costs

  By consolidating a number of physical servers onto a single or vastly smaller number of larger processors, complexity of the infrastructure is reduced, and common management tasks are concentrated and automated, which can greatly improve staff productivity.

- Improved provisioning

  Since virtual resources are abstracted from hardware and operating system issues, they are often capable of recovering much more rapidly after a crash situation. Running in this virtualized mode greatly facilitates high-speed cloning, allowing extra guests to be easily created and made available in seconds.

1.2 Sharing resources

When we say a resource is shared, this can have different meanings:

- Multiple virtual machines take turns using a resource.

  This is most obvious with sharing the CPU. z/VM dispatches virtual machines one after the other on the real processor. The required resources are provided so effectively, each virtual machine believes it *owns* the processor during its time slice. Main memory is also shared, as private pages in the working set of each virtual machine are brought in and out of main memory as needed. Again, z/VM creates the illusion that a shared resource is owned by a virtual machine. Because some work is needed to manage and allocate the resources, this type of sharing is not free. There is an overhead in z/VM for switching back and forth between virtual machines. The amount of this overhead is dependent on the number of virtual machines competing for the CPU, but on a properly tuned system, it is a relatively small part of the total available resources.

- z/VM allows virtual machines to really share memory pages.

  In this case, a portion of the virtual memory of the Linux virtual machine is mapped in such a way that multiple virtual machines point to the same page in real memory. As well as saving memory resources, this has important implications when servicing the system, as only a single change is necessary to update all of the sharing users. In z/VM, this is done through named saved systems (NSS). It is possible to load the entire Linux kernel in an NSS and have each Linux virtual machine refer to those shared pages in memory, rather than require them to have that code residing in their private working set. There is no additional cost, in terms of memory resources, for z/VM when more virtual machines start to share the NSS. Note that not all memory sharing is managed by using NSS. The virtual machines running in the z/VM layer also compete for main memory to hold their private working set. The NSS typically holds only a relatively small part of the total working set of the Linux virtual machine.

1.2.1 Overcommitting resources

It is possible to overcommit your hardware resources. When you run 16 virtual machines with a virtual machine size of 512 MB in a 4 GB z/VM system, you overcommit memory roughly with a factor of two. This works as long as the virtual machines do not require the allocated virtual storage at exactly the same time, as is very often the case.

Overcommitting resources is a good thing. We also do this in normal life. A restaurant, for example, could be seating 100 people and allow every customer to use the restrooms. The service can be provided with only a small number of restrooms. The same applies to a cinema, but that needs more restrooms per 100 people because of the expected usage pattern, which tends to be governed by the timing factors such as movie start and end. This shows how the *workload* and the time it runs affect the ability to share a hardware resource.

This example also shows that partitioning your resources (separate restrooms for male and female guests) reduces your capacity if the ratio between the two different workloads is not constant. Other requirements such as service levels might require you to do so anyway. Fortunately, the z/VM system has ample flexibility to cope well with similar practical situations within a computing system.

When the contention on the shared resources gets higher, depending on the total resources available, the chances of queuing can increase. This is one consequence of sharing that cannot be avoided. When a queue forms, the requesters are delayed in their access to the shared resources, to a variable extent. Whether such a delay is acceptable depends on service levels in place on the system, and other choices you make.

However, overcommitting is generally necessary to optimize the sharing of a resource. Big problems can arise when *all* resources in the system are being overcommitted, and are then required at the same time.

1.2.2 Estimating the required capacity

Some benchmarks deal with measuring the maximum throughput of an application. You will see references to the maximum possible number of transactions per second, the number of floating point operations per second, and the number of megabytes transferred per second for a given installed system. The maximum throughput of the system, when related to anticipated business volumes, can be a good indication of how the capacity requirements of the system should be planned. Other factors also determine how well a business application runs.

With multiple virtual machines on z/VM competing for resources, the efficiency of the application and how well it runs on the installed hardware base are very important. Typical metrics for these measurements are based upon a unit rate relation between workload and time, such as megabytes transferred per CPU second, or the number of CPU seconds required per transaction.

2

Tuning basics on System z

Each hardware and software platform has unique features and characteristics that must be taken into consideration when tuning your environment. System z processors have been enhanced over the years to provide a robust set of features and the highest level of reliability. This chapter discusses general tuning methodology concepts and explores their practical application on a System z with Linux guests. Topics covered in this chapter include processor sharing, memory, Linux guest sizing, and benchmark methodology.

© Copyright IBM Corp. 2003, 2008. All rights reserved.

2.1 The art of tuning a system

Performance analysis and tuning is a multi-step process. Regardless of which tools you choose, the best methodology for analyzing the performance of a system is to start from the outside and work your way down to the small tuning details. Start by gathering data about the overall health of systems hardware and processes. How busy is the processor during the peak periods of each day? What happens to I/O response times during those peaks? Do they remain fairly consistent, or do they elongate? Does the system get memory constrained every day, causing page waits? Can current system resources provide user response times that meet service level agreements? Following a good performance analysis process can help you answer those questions.

It is important to know what tuning tools are available and the types of information that they provide (see Chapter 3, “Linux monitoring tools” on page 13). Equally important is knowing when to use those tools and what to look for. Waiting until the telephone rings with user complaints is too late to start running tools and collecting data. How will you know what is normal for your environment and what is problematic unless you check the system activity and resource utilization regularly? Conducting regular health checks on a system also gives you utilization and performance information that can be used for capacity planning purposes.

A z/VM system offers many controls (tuning knobs) to influence the way resources are allocated to virtual machines. Very few of these controls in z/VM increase the amount of resources available. In most cases, the best that can be done is to take away resources from one virtual machine and allocate them to another one where they are better used. Whether it is wise to take resources away from one virtual machine and give them to another normally depends on the workload of these virtual machines and the importance of that work.

Tuning rarely is a one-size-fits-all approach. Instead, you will find that a system tuned for one type of workload performs poorly with another type of workload. This means that you must understand the workload that you want to run and be prepared to review your tuning when the workload changes.

2.1.1 What tuning does not do

It is important to understand that you cannot run more work than you can fit in the machine. If your System z machine has two CPUs and the workload you want to run consists of three Linux virtual machines running WebSphere, where each of them runs a CPU for 100% all day, then it will not fit. There is no z/VM tuning that will make it fit, but there might be performance issues in the applications themselves to make the workload use less than 100% all day.

When a System z machine has four CPUs and the workload consists of three Linux virtual machines that each use a CPU for 100% all day, there is little to tune. In this case, z/VM has sufficient resources to give each Linux virtual machine what it requires (though one might want to look into changes to the configuration that allow you to use all four CPUs and make things run faster).

2.1.2 Where tuning can help

Even when the different workloads add up to less than the total amount of resources available, you might find that you are still unable to run the workload with good performance. The reason for that might be that the system is short on one specific resource. In such a situation, proper tuning can make a difference. Before tuning the system and workload, you

need to understand what resource is the limiting factor in your configuration. Tuning changes tend to fall into one of these categories:

- Use less constrained resources.

  One of the benefits of running Linux systems under z/VM is the ability to share and to overcommit hardware resources. The amount of memory that Linux thinks it owns is called virtual memory. The sum of the amounts of virtual memory allocated to each Linux guest can be many times the amount of real memory available on the processor. z/VM efficiently manages memory for Linux guests. With a system that is really memory constrained, one option might be to reduce overall z/VM memory usage by reducing the virtual machine size of Linux guests.

- Get a larger share of a constrained resource.

  It is easy for users to increase the virtual memory size for Linux guests, since it is a quick change to the guest definition under z/VM. However, needlessly increasing virtual memory allocations can cause excessive paging (also known as thrashing) for the entire system. In the case of a system that is truly memory constrained, and where Linux virtual memory sizes have been assigned judiciously, consider reserving some memory pages for one particular Linux virtual machine, at the expense of all others. This can be done with a z/VM command (cp set reserved).

- Increase total available resources.

  The most obvious approach to solving memory issues is to buy more hardware. The cost of memory has declined over the years, so that may be a good option. However, additional resources can be made available by stopping unneeded utility services in the Linux systems. Be aware that tuning does not increase the total amount of system resources. It simply allows those resources to be used more effectively for critical workloads.

2.1.3 Exchange of resources

You can view tuning as the process of exchanging one resource for another. Changes to the configuration make an application use less of one resource, but more of the other. If you consider IT budget and staff hours as a resource as well, even the purchase of additional CPUs is an exchange of one resource for the other. When you tune your configuration to use less of one resource, it should not be a surprise that it will use more of another resource. Consider, for example, the WebSphere benchmark sample workload (discussed in Appendix A, “WebSphere performance benchmark sample workload” on page 191). This is a end-to-end benchmark configuration with a real-world workload driving WebSphere’s implementation of J2EE™ Web services. It runs in a multi-tiered architecture using three Linux systems. WebSphere runs in one Linux system, the IBM HTTP server in a second, and DB2® in a third Linux. Obviously, running multiple virtual machines increases some costs, because as you duplicate the Linux operating system and infrastructure, you have the additional cost of the communication between the virtual machines, and you duplicate items that otherwise could have been shared. However, by using three virtual machines, you can tune the resources given to each of these virtual machines. You can make the Web server very small so that it can be brought into memory easily and will run quickly. You can make the database virtual machine larger so that it can cache a lot of the data and thus avoid some of the I/O to disk.

By using three different virtual machines, z/VM can dispatch them on real CPUs independently. So the WebSphere server might be able to use more CPU cycles because it does not have to wait for the storage of the database server to be brought in. By just recording the number of transactions per second on an unconstrained system, you never see these subtle details. And indeed, in an unconstrained environment, it is best to make

everything as big as you can and drive it as hard as you can. In real life, however, very few of us can afford an unconstrained environment.

2.1.4 Workload profile

Real business applications have a workload profile that varies over time. The simplest form of this is a server that shows one or more peaks during the day. A more complicated workload profile is when the application is CPU intensive during part of the day, and I/O intensive during another part.

The most cost-efficient approach to running many of these types of applications is if we can adjust the capacity of the server during the day. While this might appear unrealistic, this is what we want z/VM to do. Portions of the virtual machine are brought into main memory to run. Inactive virtual machines are moved to paging space to make room.

2.2 Problem determination

Problem determination requires the skills of a systems performance detective. It is a similar process to the work of a police detective who is trying to solve a crime. In the case of IT systems performance, the crime is a performance bottleneck or a sudden worsening of response time. In the same manner as the police, the performance specialist asks questions, looks for clues, documents findings, reaches a hypothesis, tests the hypothesis by tuning or other means, and eventually solves the mystery, resulting in good system performance once again.

Where to start

Usually there is some sort of clue to get you started in the bottleneck investigation. If not, then start with simple questions that narrow the list of possibilities. Questions such as:

- What software, hardware, or application changed?
- When did the problem first occur?
- Is the performance degradation consistent, or does it come and go?
- Where is the problem occurring? Does everyone feel the pain, or only some?

These questions can be answered by IT staff or by some of the performance tools that are discussed in Chapter 3, “Linux monitoring tools” on page 13.

Steps to take

Bottleneck analysis and problem determination is greatly facilitated by sophisticated tools such as ITM Tivoli OMEGAMON® on z/VM and Linux. OMEGAMON detects performance problems and alerts you to the issue before everyone suffers under degraded response time. We use that tool for the following bottleneck analysis scenario.

Bottleneck example:

1. OMEGAMON sends out a warning alert for z/VM.

2. The OMEGAMON console contains a tree-structured list of all monitored systems and applications. It shows a warning icon (yellow triangle with an exclamation mark (!)) next to the z/VM system with the problem and next to the area of concern (Real Storage) (Figure 2-1).

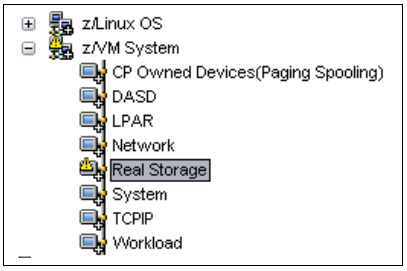


Figure 2-1 Warning icon

3. Clicking the warning icon to see additional information shows that the percentage of paging space in use is high (Figure 2-2).

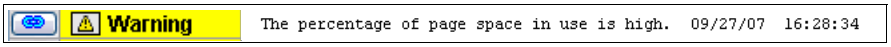


Figure 2-2 Additional warning information

4. The OMEGAMON warning message states the nature of the problem, and the link icon to the left of the warning leads the user to another panel with details about the problem and expert advice.
5. If the situation worsens, OMEGAMON sends a red alert message, saying that the page space issue is critical.

Problem determination using tools like OMEGAMON is far easier than if your only choice is to use system commands. However, the process for digging into the cause of a problem is the same no matter which tool is used.

2.3 Sizing considerations for z/VM Linux guests

The Linux memory model has profound implications for Linux guests running under z/VM.

- z/VM memory is a shared resource.

  Although aggressive caching reduces the likelihood of disk I/O in favor of memory access, the cost of caching must be considered. Cached pages in a Linux guest reduce the number of z/VM pages available to other z/VM guests.

- A large virtual memory address space requires more Linux kernel memory.

  A larger virtual memory address space requires more kernel memory for Linux memory management. When sizing the memory requirements for a Linux guest, choose the smallest memory footprint that has a minimal effect on the performance of that guest. To reduce the penalty of occasional swapping that might occur in a smaller virtual machine, use fast swap devices, as discussed in Chapter 7, “Linux swapping” on page 79.

Even though a 512 MB server does not require all that memory, it will eventually appear to use it all. Its memory cost is four times that of the 128 MB server.

More information about sizing practices for the Linux guest can be found in 5.3, “Sizing of Linux virtual memory” on page 50.

Additional information can be found in the Performance Report on the z/VM Web site at:

http://www.ibm.com/servers/eserver/zseries/zvm/perf/docs/

Additional z/VM performance tips are available on the z/VM Web site at:

http://www.ibm.com/servers/eserver/zseries/zvm/perf/tips/

2.4 Benchmarking concepts and practices

Performance is the key metric when trying to predict when a processor is out of capacity. High processor utilization is desirable from the standpoint of getting the most out of your investment, but not a true predictor for capacity planning purposes. As the workload grows on a processor, performance does not always change in a similar pattern. If a processor is running at 90% utilization and workload grows by 10%, the performance will most likely get significantly worse. A system is out of capacity when the performance becomes unacceptable and tuning efforts provide no relief.

One of the characteristics of performance is that it scales in a non-linear fashion relative to processor utilization. This means that it can remain flat or nearly constant until utilization reaches a critical point, and then performance degrades and becomes erratic. That typically happens when you pass from the linear part of a performance versus utilization chart to the exponential part of the curve. This is called hitting the knee of the curve, and that is where performance goes from mediocre to terrible in a short span of time.

Not every workload behaves the same way at any given processor utilization. Work that is designated as high priority in a system may perform fine even when total utilization is at 100%. Low priority work is the first workload that notices the burden of an overloaded system. The performance of the low-priority workload demonstrates the exponential degradation, before high-priority work shows signs of stress. As the system nears maximum capacity, workload input may continue to grow, but system throughput may either stay constant or perhaps go down. Eventually the stress of an overloaded system effects all levels of workload — low, medium, and high priorities.

Processor capacity and utilization are not the only factors affecting performance. Depending on the workload, performance is affected by all hardware components including disk, memory, and network. The operating system, software subsystems, and application code all compete for those resources. The complexity of factors affecting performance means that there is no simple way to predict what will happen when workloads change or hardware changes. One of the best ways to evaluate the effects of changes to the computing environment is to run benchmarks.

Benchmarks differ from modeling in that real hardware, software, and applications are run instead of being simulated or modeled. A good benchmark consists of many iterations testing a workload, with changes made between iterations. The changes include one or more of the following:

- Increase or decrease of the workload
- Increase, decrease, or change of hardware resources
- Adjustments to tuning parameters

After each benchmark iteration, the results are reviewed by the benchmark team before deciding on the next adjustment. Benchmarks are an excellent opportunity to try many what-if scenarios. The benchmarks run during the writing of this book are discussed in the appendixes.

3

Linux monitoring tools

The purpose of this book is to discuss the analysis of z/VM and Linux performance monitor data, rather than providing a comparison of monitoring tools. However, an overview of the tools used in writing this book is provided in this chapter as background information. It is important to understand the types of tools available for the System z environment and the value they provide to a systems administrator or performance specialist.

In this chapter, we start by describing the performance data produced by some of the IBM z/VM and Linux performance monitoring software products. These products can go beyond simple monitoring, with charting capabilities, problem diagnosis and alerting, and integration with other software for monitoring of other systems such as z/OS, databases, and Web servers. The real-time monitoring tools used in writing this book include the VM Performance Toolkit, and IBM Tivoli OMEGAMON XE for z/VM and Linux. The historical post-processing software products include the VM Performance Reporting Facility and the VM Performance Analysis Facility. Following the sections on performance tuning software products, are discussions on simple tools or commands that are either provided with the operating systems or are available to the user community as free-ware.

In each section, the performance tuning tools are discussed in the context of the steps involved in analyzing the performance of a system.

© Copyright IBM Corp. 2003, 2008. All rights reserved.

3.1 Introduction to system performance tuning tools

Sometimes the hardest part of the process to improve system performance is deciding how to get started, or what tool to use. Real-time monitors and tools provide information useful in solving today's performance problem. Historical reporting tools give information that can be useful for health checks, problem determination, and trend analyses. Sophisticated monitoring software can also proactively check for performance issues and automatically act to prevent severe outages.

Real-time monitors can be as simple as Linux or z/VM commands that are built in to each system. These commands might be your first options as tools for performance analysis if more robust software products are not available. IBM software products such as the z/VM Performance Toolkit and Tivoli OMEGAMON XE on z/VM and Linux are also examples of real-time monitoring tools. They were used in combination during our benchmarks to capture performance information. In this book, the IBM Tivoli OMEGAMON suite of software is also referred to as OMEGAMON. The z/VM Performance Toolkit and OMEGAMON provide detailed system performance statistics in tailorable tabular and graphic formats. They also have autonomic computing capabilities that respond to problem situations without operator intervention.

Historical reporting is outside the scope of most system commands. Therefore, health checks, problem determination, and trend analysis are best done by software that collects the data for later analysis. After the data is collected, reporting tools can filter the massive amount of data, helping to narrow the search for relevant information. The historical reporting tools that were chosen for this book include IBM VM Performance Reporting Facility and the VM Performance Analysis Facility.

3.2 IBM z/VM Performance Toolkit

The Performance Toolkit for VM (also called the Performance Toolkit) is designed to assist operators and system programmers or analysts in the following areas:

- System console operation in full panel mode

  The features provided have been designed to facilitate the operation of VM systems, thereby improving operator efficiency and productivity.

- Performance monitoring on z/VM systems

  An enhanced real-time performance monitor allows system programmers to monitor system performance and to analyze bottlenecks. Features included in the monitor, and the manner in which data is displayed, have both been designed to improve the system programmer's productivity when analyzing the system, and to allow even a new user to work efficiently with the tool. The Performance Toolkit can help system programmers to make more efficient use of system resources, increase system productivity, and improve user satisfaction.

The z/VM Performance Toolkit features include:

- Automatic threshold monitoring of many key performance indicators with operator notification if user-defined limits are exceeded
- Special performance monitoring mode with displays for monitoring:
  - General CPU performance
  - System and user storage utilization and management

- Channel and I/O device performance, including cache data
- Detailed I/O device performance, including information about the I/O load caused by specific minidisks on a real disk pack
- General user data: resource consumption, paging information, IUCV and VMCF communications, wait states, response times
- Detailed user performance, including status and load of virtual devices
- Summary and detailed information about shared file system servers
- Configuration and performance information for TCP/IP servers
- Linux performance data – system execution space (SXS) performance data
- Virtual networking configuration and performance data

The collected performance data is also used to provide:

- A re-display facility for many of the key figures shown on the general CPU performance and storage utilization panels, which allows browsing through the last measurements (up to twelve hours)
- Graphical history plots with a selection of up to four re-display variables, either as simple plots, or, if the Graphical Data Display Manager program (GDDM®, 5684-007 or 5684-168) is available, also in the form of GDDM graphics
- Graphical variable correlation plots (simple plots or GDDM graphics) that show how the values of specific variables are correlated to the values of another variable

> **Note:** GDDM is required for generating graphics on the console of a virtual machine. However, no additional software is required when generating graphics with Web browsers using the WWW interface.

- For later reference, the accumulated performance data can also be used for creating:
  - Printed performance reports
  - Simple performance history files
  - Extended trend files
- For capacity planning, the history files on disk can be used for:
  - Performance trend graphics
  - Numerical history data analysis

What we now call the VM Performance Toolkit can trace its origins back seventeen years, and it has evolved from an operations console with a few performance commands, to a highly functional, varied coverage, highly detailed reporting and real-time monitoring tool. Communication with, and subsequent collection, of data from a Linux guest allows it to produce a consolidated view of the hardware, LPAR, VM, and Linux levels. The user interface to the Performance Toolkit is menu driven, but you can fastpath by entering the full or abbreviated report name to go directly to the one you need. Once inside a report, many of them have sub menus, or further navigation is possible by placing the cursor under the variable and pressing the Enter key. Indeed, a lot of reports contain many layers of navigation, for example, you can look at an ordered list of DASD volumes, and see the overall activity, plus controller functions. The you can go deeper, and look at a given volume in more detail, then go down into further detail and look at the minidisk device activity. This type of navigation functionality greatly facilitates problem determination, as well as making the overall understanding of your system, even down at deep level, much easier because of this

enhanced accessibility. The various data reports are divided into the groups of general system data, I/O data, history data, and user data. The top portion of the menu is shown in Figure 3-1.

```
FCX124          Performance Screen Selection  (FL530        )    Perf. Monitor

General System Data        I/O Data                    History Data (by Time)
1. CPU load and trans.     11. Channel load            31. Graphics selection
2. Storage utilization     12. Control units           32. History data files*
3. Reserved                13. I/O device load*        33. Benchmark displays*
4. Priv. operations        14. CP owned disks*         34. Correlation coeff.
5. System counters         15. Cache extend. func.*    35. System summary*
6. CP IUCV services        16. DASD I/O assist         36. Auxiliary storage
7. SPOOL file display*     17. DASD seek distance*     37. CP communications*
8. LPAR data               18. I/O prior. queueing*    38. DASD load
9. Shared segments         19. I/O configuration       39. Minidisk cache*
A. Shared data spaces      1A. I/O config. changes     3A. Storage mgmt. data*
B. Virt. disks in stor.                                3B. Proc. load & config*
C. Transact. statistics    User Data                   3C. Logical part. load
D. Monitor data            21. User resource usage*    3D. Response time (all)*
E. Monitor settings        22. User paging load*       3E. RSK data menu*
```

Figure 3-1 z/VM Performance Toolkit menu

A good place to start looking at information is in the categories of general system data and I/O data. Starting with the top options, begin a performance health check of a system by selecting option **1. CPU load and trans**. This option shows overall processor utilization, system contention, and information about the user population. The user data includes the total number of users, number of active users, and the number of users who are waiting for a resource (storage pages, I/O). The display shown in Figure 3-2 is produced.

```
FCX100      CPU 2094  SER 2991E  Interval 13:10:56 - 13:11:56    Perf. Monitor

CPU Load                                                 Vector Facility   Status or
PROC TYPE %CPU  %CP %EMU %WT %SYS %SP %SIC %LOGLD  %VTOT %VEMU  REST ded. User
P00  CP      1    0    1  99    0   0   94      1     ..    ..   ... Master
P01  CP      1    0    1  99    0   0   95      1     ..    ..   ... Alternate
P02  CP      1    0    1  99    0   0   91      1     ..    ..   ... Alternate
P03  CP      1    0    1  99    0   0   93      1     ..    ..   ... Alternate

Total SSCH/RSCH      6/s     Page rate       1.8/s     Priv. instruct.    17/s
Virtual I/O rate     5/s     XSTORE paging  13.0/s     Diagnose instr.     6/s
Total rel. SHARE     400     Tot. abs SHARE    10%

Queue Statistics:     Q0     Q1     Q2     Q3     User Status:
VMDBKs in queue        2      2      0      2     # of logged on users      19
VMDBKs loading         0      0      0      0     # of dialed users          0
Eligible VMDBKs               0      0      0     # of active users         12
El. VMDBKs loading            0      0      0     # of in-queue users        6
Tot. WS (pages)   124746 117490      0 882081     % in-Q users in PGWAIT     0
Expansion factor              0      0      0     % in-Q users in IOWAIT     0
85% elapsed time   2.408   .301  2.408  14.45     % elig. (resource wait)    0
```

Figure 3-2 z/VM Performance Toolkit CPU display

The next step in analyzing the system is to look into another area of the Performance Toolkit menu, depending on what was found in the CPU display. The example shown in Figure 3-2 on page 16 is for a lightly loaded system with no performance issues. But, if that display had shown high CPU utilization numbers, then the next step is to look at the main menu category of user data. The user data display shown in Figure 3-3 gives more details on which users are consuming the most processor resources.

```
FCX112      CPU 2094  SER 2991E  Interval 13:12:56 - 13:13:56    Perf. Monitor
.          ____     .     .     .     .     .     .     .     .     .
           <----- CPU Load -----> <------ Virtual IO/s ------>
                <-Seconds->   T/V
Userid     %CPU  TCPU  VCPU Ratio Total DASD Avoid Diag98   UR Pg/s  User Status
>>Mean>>    .20  .117  .113   1.0    .2   .2    .0     .0   .0   .0  ---,---,---
LNXDB2     1.48  .888  .864   1.0   1.1  1.1    .0     .0   .0   .0  EME,CL0,DIS
LNXSU1     1.48  .885  .853   1.0   1.0  1.0    .0     .0   .0   .0  ESA,CL3,DIS
LNXWAS      .55  .332  .312   1.1    .7   .7    .0     .0   .0   .0  EME,CL3,DIS
LNXIHS      .10  .058  .055   1.1    .6   .6    .0     .0   .0   .0  EME,---,DOR
LNXSU4      .07  .039  .034   1.1    .4   .4    .0     .0   .0   .0  EME,CL1,DIS
PERFSVM     .03  .018  .017   1.1    .4   .2    .1     .0   .0   .0  ESA,---,DOR
TCPIP       .01  .004  .003   1.3    .0   .0    .0     .0   .0   .0  ESA,---,DOR
DIRMAINT      0     0     0   ...     0    0     0      0    0    0  ESA,---,DOR
DISKACNT      0     0     0   ...     0    0     0      0    0    0  ESA,---,DOR
DTCVSW1     .00  .000  .000   ...    .0   .0    .0     .0   .0   .0  ESA,---,DOR
DTCVSW2     .00  .000  .000   ...    .0   .0    .0     .0   .0   .0  ESA,---,DOR
GCS           0     0     0   ...     0    0     0      0    0    0  ESA,---,DOR
OPERSYMP      0     0     0   ...     0    0     0      0    0    0  ESA,---,DOR
PVM         .00  .000  .000   ...    .1   .0    .0     .0   .0   .0  ESA,---,DOR
RSCS        .00  .002  .001   2.0    .3   .0    .0     .0   .0   .0  ESA,---,DOR
Select a user for user details or IDLEUSER for a list of idle users
```

Figure 3-3 z/VM Performance Toolkit User display

The z/VM Performance Toolkit provides detailed reports on all aspects of system and user performance information to assist with system health checks or problem determination into a live situation.

More information about how to use the Performance Toolkit for problem determination can be found in *Linux on IBM eServer zSeries and S/390: Performance Toolkit for VM*, SG24-6059.

3.3 IBM z/VM Performance Reporting Facility

This product, used in the production of some performance reports in this book, is no longer available for ordering.

VM Performance Reporting Facility (VMPRF 5684-073) provides performance management capabilities for VM systems. It is a post-processor of VM/ESA® or z/VM CP monitor data. It is a performance analysis tool for VM systems that can be used to detect and diagnose performance problems, analyze system performance, and provide printed reports that show the utilizations and response times of key system components. History files created by VMPRF can be used as input for performance analysis by VMPAF, Tivoli Performance Reporter for OS/390® (5665-397), or installation-written application programs. VMPRF is useful for systems analysts, systems programmers, and systems administrators. It simplifies the work of resource management in a VM/ESA or z/VM system.

VMPRF simplifies performance analysis and resource management of z/VM. Reports and history files produced by VMPRF include information about:

- System resource utilization, transaction response time, and throughput
- Resource utilization by the user ID
- DASD activity and channel utilization

VMPRF runs as a post-processor on previously collected data. Parameters input to the program execution specify the types of reports desired and time frame for data selection. The summary report gives very detailed statistics over a wide range of parameters at hardware, operating system, and user level.

```
PRF002  Run 01/12/1991 17:38:31        SYSTEM_SUMMARY_BY_TIME                                                    Page    1
                                       System Performance Summary by Time
From 11/02/1989 10:01:02                                                                          KGNVMC
To   11/02/1989 11:00:02                                                                          CPU 4381      SN  17009
For    3540 Secs 00:59:00              SP 2 test data                                             VM/XA SP 21.00 SLU 0000
______________________________________________________________________________________________________________________

          <--------CPU---------> <Vec> <--Users--> <---I/O---> <Stg> <-Paging-> <Spl> <----------UP+MP Transactions---------->
                <--Ratio-->                                           <--Rate-->       <--Response Time--> <----Throughput---->

                                                        DASD        PGIN   Rd
From  To      Pct        Cap-  On- Pct     Log-                     Resp         and  and                  Non-  Quick         Non-  Quick
Time  Time   Busy   T/V  ture line Busy     ged Activ Rate   Time Elist PGOUT   Wr Rate   Triv   Triv   Disp   Triv   Triv   Disp

10:01 10:05  57.7  1.19 .8697  2.0     0    36    16   62   19.1   0.2    34   51    1  0.131  3.631  0.440   0.87   0.27   0.08
10:05 10:10  57.5  1.28 .8754  2.0     0    36    16   78   16.3     0    24   26    1  0.143  2.036  0.521   1.05   0.33   0.11
10:10 10:15  56.0  1.30 .8567  2.0     0    37    17   79   18.3     0    35   51    1  0.158  2.801  0.705   1.01   0.35   0.15
10:15 10:20  57.1  1.29 .8607  2.0     0    37    17   80   17.5   0.8    54   65    1  0.212  1.604  0.714   0.80   0.32   0.19
10:20 10:25  55.7  1.29 .8700  2.0     0    37    17   88   16.6     0    34   37    1  0.206  6.509  0.543   0.92   0.35   0.12
10:25 10:30  60.8  1.31 .8697  2.0     0    37    19   75   18.1   0.4    39   59    2  0.198 42.132 192.42   1.02   0.49   0.23
10:30 10:35  51.5  1.42 .8283  2.0     0    37    18   90   20.7   0.2    46   91    1  0.238  3.719  0.545   0.85   0.44   0.20
10:35 10:40  58.9  1.25 .8540  2.0     0    37    18   77   24.3   1.4    60  123    2  0.286  5.317  0.691   0.80   0.29   0.28
10:40 10:45  79.7  1.26 .8693  2.0     0    37    19   83   23.4   1.2    53   91    3  0.257 25.623 12.827   0.78   0.35   0.30
10:45 10:50  92.2  1.15 .9107  2.0     0    37    18   62   24.7   0.2    45   67    1  0.251 19.563  0.734   0.94   0.26   0.16
10:50 10:55  83.3  1.16 .9049  2.0     0    36    16   61   23.4   1.0    41   38    2  0.209 11.458 19.341   0.87   0.29   0.09
10:55 11:00  31.0  1.20 .8256  2.0     0    36    16   32   17.4   3.4    10    9    1  0.216 17.327  0.569   0.90   0.26   0.05

10:01 11:00  61.9  1.25 .8716  2.0   0.0    37    17   72   20.0   0.7    40   59    1  0.207 12.895 25.678   0.90   0.33   0.17
```

Figure 3-4   VMPRF summary report

After reviewing the overall statistics in the system summary by time report, select the report that provides details on your area of interest. Figure 3-5 is a sample report for CPU Utilization.

```
PRF003  Run 12/10/1990 09:54:55        PROCESSORS_BY_TIME                                                    Page    1
                                       Activity for Each Processor by Time
From 07/28/1988 10:35:35                                                                          YOURSYST
To   07/28/1988 11:04:35                                                                          CPU 3090      SN  72512
For    1740 Secs 00:29:00              Your System Description                                    VM/XA SP 02.00 SLU 0000
______________________________________________________________________________________________________________________

                <--------Percent Busy-------> <-----------Rate----------> <--------PLDV--------> <Avl> <-----Paging----> <-VM->
                                                                              <-----VMDBKs-----> <Lst>                   <Comm>
                                                                              <Ct> <----Rate--->
            C                                                      SSCH Pct Mean         Moved  Secs         Pct Page
From  To    P                                  Inst                 and Em- when            to   bet  PGIN Fast- Read     Msg
Time  Time  U Total  User  Syst  Emul  Vect    Siml    DIAG   SIGP  RSCH pty Non0 Stolen Master Empty  Rate  path Rate    Rate

10:35 10:39 1  53.1  40.1  13.0  14.8     0     344     252      0   105   0 3422   94.4  549.5    48   127  92.3   16      31
10:39 10:44 1  57.8  44.0  13.8  17.0     0     392     268      0   117   0 2801  112.1  570.9    75   138  92.1   15      37
10:44 10:49 1  69.5  55.6  14.0  25.9     0     449     358      0   137   0 3034  159.2  671.2   100   159  92.8   14      32
10:49 10:54 1  85.1  70.4  14.7  31.1     0     634     556      0   163   0 2643  205.1 1006.7    19   138  91.0   11      30
10:54 10:59 1  93.7  77.4  16.3  30.6     0     771     704      0   170   0 3638  210.4 1261.2    75   144  90.2   11      36
10:59 11:04 1  98.8  83.0  15.8  44.6     0     605     525      0   182   0 2495  196.1  900.0    33   161  91.2   11      36

10:35 10:39 2  35.3  32.9   2.3  25.4     0     259     159      0    92   0 3011   84.1      0   120    84  96.7    9       5
10:39 10:44 2  42.1  39.4   2.8  31.2     0     742     188      0   108   0 3664  110.4      0   300    96  96.5    9       5
10:44 10:49 2  61.5  58.3   3.2  49.4     0     307     216      0   133   0 3021  171.0      0   150   106  97.1   10       5
10:49 10:54 2  77.1  72.9   4.2  62.0     0     413     318      0   160   0  943  235.7      0    25   128  97.0   10       6
10:54 10:59 2  87.1  82.6   4.5  70.8     0     501     402      0   154   0  743  257.9      0    60   161  97.4   12       6
10:59 11:04 2  98.7  94.0   4.7  82.3     0     418     332      0   174   0  249  228.0      0    43   135  97.0   12       5
```

Figure 3-5   VMPRF CPU utilization report

The VM Performance Reporting Facility is easy to customize and offers very detailed reports for all aspects of systems performance. These reports can be run for specified time frames to show changes in systems performance over the day.

3.4 IBM z/VM Performance Analysis Facility

This product, used in the production of some performance reports in this book, is no longer available for ordering.

IBM Performance Analysis Facility/VM (VMPAF 5684-130) aids an analyst in performance problem determination, tuning, and trend analysis. It applies interactive graphics and statistical correlations to performance history data produced by the following IBM products: VMPRF and the VM Performance Toolkit. VMPAF also allows you to view data from multiple systems. This can be used to compare data from the same system before and after changes have been made.

The interactive graphics environment shows how performance variables behave over time. It also provides statistical information about these variables, including minimum, maximum, and average values for the time period. The program calculates the coefficient of correlation between each of these variables and a coefficient of variance representing the relative variability of the data parameters under investigation.

The VMPAF chart shown in Figure 3-6 was created to view the correlation of virtual I/O to swap I/O after benchmark testing of various swapping devices.

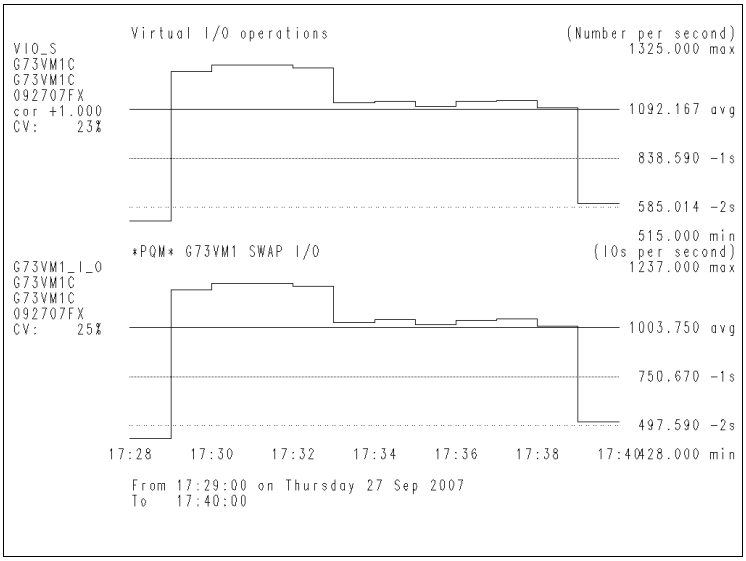


Figure 3-6 VMPAF chart comparing virtual I/O and swap I/O

3.5 IBM Tivoli OMEGAMON XE on z/VM and Linux

Tivoli OMEGAMON XE on z/VM and Linux is one of a suite of Tivoli Management Services products called Tivoli Enterprise Monitoring Agents. These products use common shared technology components to monitor your mainframe and distributed systems on a variety of platforms, and to provide workstation-based reports that you can use to track trends and understand and troubleshoot system problems.

Tivoli OMEGAMON XE on z/VM and Linux, V4.1.0 is a powerful product that gives you the ability to view data collected from multiple systems on one easy-to-use, flexible interface. With this monitoring agent, you can view z/VM data obtained from the Performance Toolkit for VM (also called the Performance Toolkit). You can also display Linux on System z

performance data. This dual capability allows you to solve problems quickly and helps you to better and more easily manage a complex environment.

OMEGAMON monitoring software provides a portal that is accessible via a desktop client or a browser. This portal interface allows users to monitor systems without having to be logged onto either Linux or z/VM. The portal provides a single point of control for monitoring all of you Linux and z/VM images as well as z/OS and its subsystems, databases, Web servers, network services, and WebSphere MQ.

The OMEGAMON portal displays the following types of data:

- Memory usage, disk input and output (I/O) usage, and paging activity by workload name
- System-wide data by resource, such as logical partition usage and paging data
- System-wide spooling space usage
- TCP/IP network statistics for enabled network server virtual machines
- HiperSockets™ utilization data
- z/VM CPU data
- Linux on IBM System z workload statistics

The beginning phase of checking the performance of a system, or multiple systems, using the OMEGAMON product is to look at the high-level displays for each system. In our environment we are only concerned with the performance of z/VM and its Linux guests. Since z/VM controls all of the hardware resources and virtualizes that environment for the Linux guests, the OMEGAMON display for z/VM is a good starting point. The following series of graphs from the OMEGAMON display highlight the key performance indicators of CPU, disk, and storage utilizations.

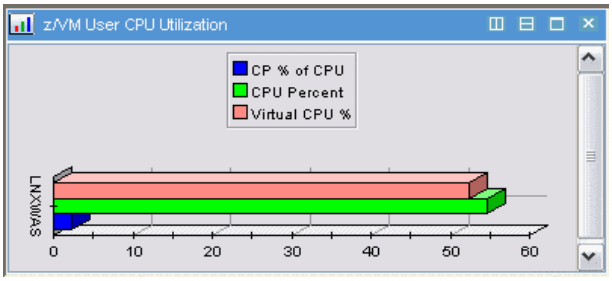


Figure 3-7 OMEGAMON CPU utilization graph

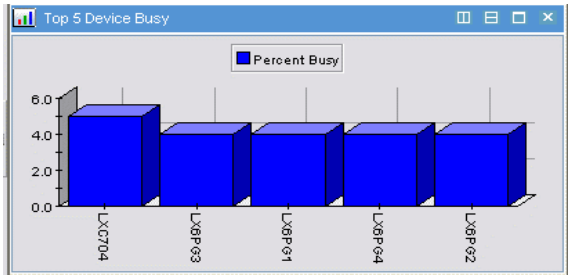


Figure 3-8 OMEGAMON device utilization graph

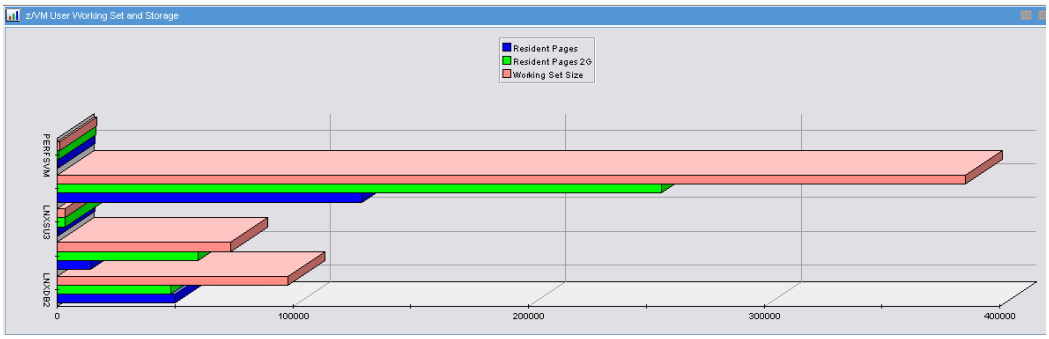


Figure 3-9 OMEGAMON storage utilization graph

The preceding graphs depict system utilization in graphic form. The same data can be displayed from the OMEGAMON portal in text or tabular form through options in the panels. In a System z environment it is possible to share processors between LPARS, which means that the processors assigned to this z/VM system might be shared by other z/VM, Linux, or z/OS systems. This means that an investigation of system performance must include a view of the total hardware complex. The graph shown in Figure 3-10 from OMEGAMON has the data.

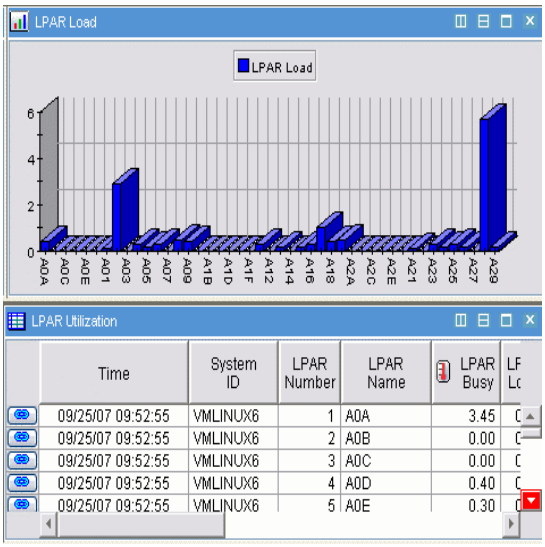


Figure 3-10 OMEGAMON LPAR utilization graph

The LPAR graphic shown in Figure 3-10 has a bar chart showing processor utilization of every LPAR on the System z, and it has a table with detailed information about each LPAR. The table has scroll bars to expand the table and show more columns and rows. On the left of each row is a link icon, which connects a user to additional information relating to that row (or LPAR).

After careful inspection of the system performance information available via the z/VM displays from OMEGAMON, the next step in a performance evaluation is to look at statistics from inside one or many of the Linux guests that are running on z/VM.

The overview workspace for Linux guests displays many of the key performance indicators. In the Figure 3-11, OMEGAMON shows the top Linux systems and their CPU utilization, their load on the system for the last 15 minutes, and a table of system statistics (context switches, paging rates, length of time that the system has been up, and number of users logged on).

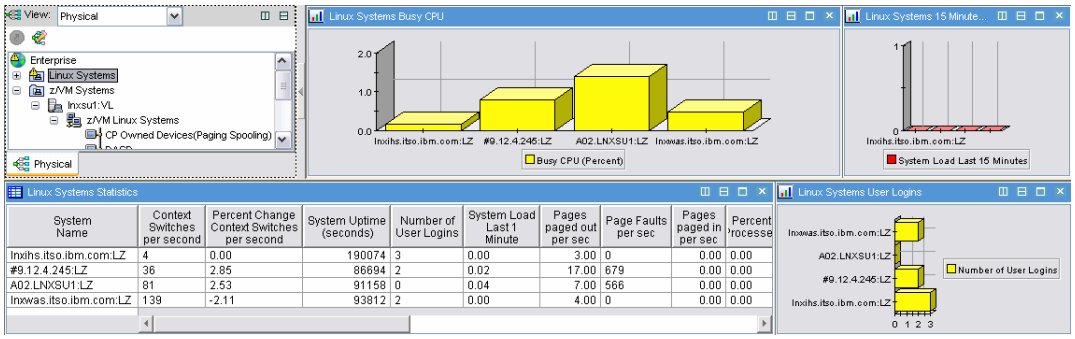


Figure 3-11 OMEGAMON Linux guest performance overall view

After reviewing the high-level information that is available on the collective Linux display, additional OMEGAMON panels show details on Linux systems, processes, files, and networks. Some of those panels are shown below.

Figure 3-12 OMEGAMON Linux system graph

Linux Guest Appl Data

Time	System ID	LPAR Name	User ID	Virtual CPUs	Total CPU	User CPU	Kernel CPU	Nice CPU	Percent IRQ	Percent Soft IRQs	Percent I/O Wait	Percent CPU Idle	Runnable Processes	Processes Waiting for I/O	Total Processes	Avg P Las
09/25/07 11:25:56	VMLINUX6	A02	LNXDB2	1	15.20	9.00	5.60	0.00	0.30	0.30	2.80	81.30	5	1	194	
09/25/07 11:25:56	VMLINUX6	A02	LNXIHS	1	1.70	0.90	0.40	0.00	0.20	0.30	0.10	98.20	2	0	160	
09/25/07 11:25:56	VMLINUX6	A02	LNXSU1	1	2.20	1.79	0.40	0.00	0.00	0.00	0.00	97.80	2	0	286	
09/25/07 11:25:56	VMLINUX6	A02	LNXWAS	1	5.30	5.00	0.20	0.00	0.00	0.00	0.50	94.00	30	0	202	

Figure 3-13 OMEGMON Linux application table

Process Information Detail

Process Command name (Unicode)	Process ID	Process Parent ID	Process State	Process System CPU (Percent)	Process User CPU (Percent)	Cumulative Process System CPU (Percent)	Cumulative Process User CPU (Percent)	Kernel Priority	Nice Value	Total Size(pages)	Resident Set Size(pages)	M
cupsd	1244	1	Sleeping	594.07	0.05	0.00	0.00	16	0	2300	1091	
portmap	1231	1	Sleeping	594.07	0.00	0.00	0.00	15	0	462	122	
cron	1240	1	Sleeping	594.07	0.00	0.00	0.00	16	0	558	212	
gengat	1206	1	Sleeping	969.02	0.02	0.00	0.00	16	0	871	251	

Figure 3-14 OMEGAMON Linux processes table

Figure 3-14 shows the status and resource utilization for each process. Tables such as this in OMEGAMON can be filtered (to eliminate meaningless data) and sorted (to place date of high interest at the top).

With the Tivoli OMEGAMON XE for z/VM and Linux agent, you can also perform the following tasks:

- Navigate to greater levels of detailed performance data. For Linux guests, this monitoring agent provides links to Tivoli Monitoring Agent for Linux OS workspaces directly from the Tivoli OMEGAMON XE on z/VM and Linux workspaces.
- Connect to the z/VM host system by means of TCP/IP to access the Performance Toolkit panels.
- View expert advice on how to identify and resolve performance problems.

IBM Tivoli OMEGAMON comes with default thresholds for key system performance indicators. When a threshold is exceeded, a situation alert is generated and an automated action is taken to address the problem. The situation console log workspace shows the active situation alerts. In Figure 3-15, the upper panel lists open situations. The lower left panel has the situation count for the last 24 hours. The lower right panel lists the situation events that have been acknowledged by the current user.

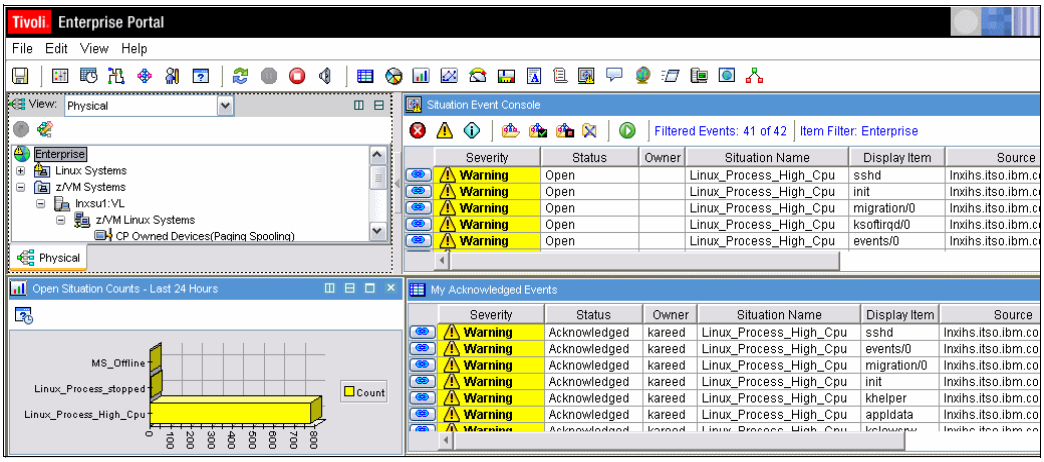


Figure 3-15 OMEGAMON situation alerts

Default workspaces and situations enable you to start monitoring your enterprise as soon as the Tivoli OMEGAMON XE on z/VM and Linux software is installed and configured. The user interface supports several formats for viewing data, such as graphs, bar charts, and tables. Workspaces and situations can be customized to meet the needs of your enterprise.

3.6 Linux system tools

Sometimes the hardest part of the process to improve system performance is deciding how to get started or what tool to use. The simple commands described in the next few pages can be used as the initial step in evaluating system performance. These commands are either standard parts of the Linux operating system or are available as free-ware on the Internet.

The vmstat command

The **vmstat** command displays current statistics for processes, usage of real and virtual memory, paging, block I/O, and CPU. The left-most columns show the number of running processes and number of blocked processes. The memory section shows memory being swapped out, free memory, the amount of buffer containing inodes and file metadata, and cached memory for files being read from disk. The swap section lists swaps in and swaps out. The I/O section reports the number (kb) of blocks read in and written out. System in and cs

represent interrupts and context switches per second. The CPU section headers of us, sy, id, and wa represent percentage of CPU time count on users, system, idle, I/O wait, and steal, respectively. The **vmstat** command is useful in identifying memory shortages and I/O wait issues, in addition to situations where virtual CPUs are not backed up by physical CPUs.

Figure 3-16 lists system information and adds an update line three times, at five-second intervals.

```
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0     68  47632  77916 160924    0    0     3    12  141   44  1  0 99  0  0
 0  0     68  47648  77920 160920    0    0     0    24   18   31  0  0 100  0  0
 0  0     68  47260  77928 160912    0    0     0     4 1879   42  1  0 98  0  0
```

Figure 3-16 Linux command - vmstat

Top command

The **top** command displays process statistics and a list of the top CPU-using Linux processes. Options are available to interactively change the format and content of the display. The process list may be sorted by run time, memory usage, and processor usage. Top shows the total number of processes, the number running, and sleeping. Information is updated every three seconds, or at a chosen interval. It is useful in obtaining a snapshot of the processes that are running on a Linux system and their CPU and memory consumption.

Figure 3-17 shows an interactive display of system information.

```
top - 14:47:33 up 20:41,  2 users,  load average: 0.05, 0.03, 0.00
Tasks:  92 total,   2 running,  90 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.2%us,  0.2%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    505168k total,   457708k used,    47460k free,    77804k buffers
Swap:   719896k total,       68k used,   719828k free,   160520k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20090 root      16   0  2640 1300  980 R    0  0.3   0:00.07 top
    1 root      16   0   848  316  264 S    0  0.1   0:01.36 init
    2 root      RT   0     0    0    0 S    0  0.0   0:00.04 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.01 ksoftirqd/0
    4 root      RT   0     0    0    0 S    0  0.0   0:00.04 migration/1
```

Figure 3-17 Linux command - top

Process status (ps) command

The **ps** command displays information about running processes. It is a one-time display (no automatic updating) that includes the UID, process ID, start date and time, and process name. There are many options available with the **ps** command. Useful information about parameters and display format can be found via the command **man ps**.

Figure 3-18 shows the **ps** command with the option efH for listing every process, full format, hierarchy structure.

```
lnxdb2:/opt/IBM/ITM/bin # ps -efH
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Sep25 ?        00:00:01 init [5]
root         2     1  0 Sep25 ?        00:00:00   [migration/0]
root         3     1  0 Sep25 ?        00:00:00   [ksoftirqd/0]
root         4     1  0 Sep25 ?        00:00:00   [migration/1]
root         5     1  0 Sep25 ?        00:00:00   [ksoftirqd/1]
root         6     1  0 Sep25 ?        00:00:05   [events/0]
root         7     1  0 Sep25 ?        00:00:05   [events/1]
root         8     1  0 Sep25 ?        00:00:00   [khelper]
root         9     1  0 Sep25 ?        00:00:00   [kthread]
root        12     9  0 Sep25 ?        00:00:00     [kblockd/0]
root        13     9  0 Sep25 ?        00:00:00     [kblockd/1]
root        31     9  0 Sep25 ?        00:00:00     [cio]
```

Figure 3-18 Linux command - ps

System status (sysstat) tool

This package consists of several Linux tools to collect system data. The sysstat package is a widely used Linux standard tool. It is included in your distribution or can be downloaded from the Web:

http://pagesperso-orange.fr/sebastien.godard/

If you install the source package you have to compile the package on Linux on System z to use it. The sysstat package consists of the following components:

- **sadc** data gatherer - stores data in binary file
- **sar** reporting tool - reads binary file created with 'sadc' and converts it to readable output
- **mpstat** - processor utilization
- **iostat** - I/O utilization

We use sar, which prints the following information:

- Process creation
- Context switching
- All/single CPU utilization
- Network utilization
- Disk statistics

For a more detailed description go to:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_how_tools.html#sysstat

OProfile tool

OProfile is an open source profiler for Linux systems. It offers profiling of all running code on a Linux system, including the kernel, shared libraries, application binaries, and so on, and provides a variety of statistics at a low overhead (varying from 1–8%) depending on the workload. It is released under the GNU GPL. OProfiler consists of a kernel driver, a daemon for collecting sample data, and tools for report generation.

OProfile generally supports Linux kernel 2.2, 2.4, 2.6, and later. But, for Linux on System z, OProfile only supports kernel 2.6 or later. For SUSE Linux, the kernel level must be at least kernel-s390(x)-2.6.5-7.191, which starts from SLES9 SP2. For RedHat Linux, the kernel level must be at least kernel-2.6.9-22.EL, which is RHEL4 U2.

OProfile utilizes CPU's performance counters to count events for all of the running code, and aggregates the information into profiles for each binary image. However, System z hardware currently does not have support for this kind of hardware performance counters utilized by OProfile, so the timer interrupt is used instead.

Example 3-1 illustrates a typical usage of OProfile on a Linux guest running on z/VM. To use the OProfile tool, the timer should be turned on by using command **sysctl**. After all the profiling is done, turn the timer off. To set up the environment, use the **opcontrol** command to tell OProfile the location of the vmlinux file corresponding to the running kernel. Then it is ready to collect the profile data. After the test and tool shutdown, the profiling report can be generated. Figure 3-19 on page 26 illustrates an output example from OProfile.

Example 3-1 How to use OProfile on Linux on System z

```
sysctl -w kernel.hz_timer=1                          1
gunzip /boot/vmlinux-2.6.5-7.201-s390x.gz            2

opcontrol --vmlinux=/boot/vmlinux-2.6.5-7.201-s390x  3
opcontrol --start                                    4
```

```
<DO THE TEST>

opcontrol --shutdown                                    5
opreport                                                6
```

Note the following points for the numbers in black:

1 The command to turn on the timer. When the profiling is finished, specify it to be 0 to turn off the timer.

2 Unzip the vmlinux file if none exists.

3 Specify the vmlinux file for the running kernel.

4 Start the OProfile deamon, and begin profiling.

5 Stop the OProfile deamon, and stop profiling.

6 Product symbol or binary image summaries.

In Figure 3-19, the CPU information is not correctly displayed. The System z hardware does not support the CPU performance counters required by OProfile yet. The output includes:

- *vma* (Virtual Memory Area) is a contiguous area of virtual address space. These areas are created during the life of the process when the program attempts to memory map a file, links to a shared memory segment, or allocates heap space.
- *samples* is the number of samples for the symbol.
- *%* means the percentage of samples for this symbol relative to the overall samples for the executable.
- *app name* is the application name to which the symbol belongs.
- *symbol name* is name of the symbol that was executed.

From the report of OProfile, we can analyze the performance of a target module and adjust tuning accordingly. For example, from Figure 3-19, three modules of the application mcf_bas.z_Linux consume most of the CPU time. To improve the performance of this application, more attention should be paid to these three modules than the others.

```
CPU: CPU with timer interrupt, speed 0 MHz (estimated)
Profiling through timer interrupt
vma        samples  %        app name          symbol name
80002840   5862     34.8970  mcf_base.z_Linux  price_out_impl
800012c8   5221     31.0811  mcf_base.z_Linux  refresh_potential
80003cb4   4398     26.1817  mcf_base.z_Linux  primal_bea_mpp
80003b60   408      2.4289   mcf_base.z_Linux  sort_basket
0001a67c   345      2.0538   vmlinux           default_idle
800013d8   138      0.8215   mcf_base.z_Linux  flow_cost
800033bc   98       0.5834   mcf_base.z_Linux  update_tree
800020f8   88       0.5239   mcf_base.z_Linux  dual_feasible
800036a4   72       0.4286   mcf_base.z_Linux  primal_iminus
8000323c   40       0.2381   mcf_base.z_Linux  write_circulations
80002720   24       0.1429   mcf_base.z_Linux  insert_new_arc
```

Figure 3-19 An output example from OProfile

For more detail and documentation of the OProfile commands see:

http://oprofile.sourceforge.net/docs/

3.7 z/VM system tools

Like Linux, the z/VM operating system has commands to display system performance information. These commands are control program (CP) system commands. Entering the command qualifier of CP is not required. z/VM responds the same to **cp indicate**, as **indicate**.

cp indicate

The **cp indicate** command displays information about overall processor utilization (or LPAR processor utilization), paging rate, mini-disk cache rate, and user queues. Additional information about the z/VM **indicate** command can be found at the IBM Redbooks Web site (ref tips0592.html).

```
16:39:36 INDICATE
16:39:36 AVGPROC-001% 04
16:39:36 XSTORE-000019/SEC MIGRATE-0001/SEC
16:39:36 MDC READS-000001/SEC WRITES-000001/SEC HIT RATIO-100%
16:39:36 PAGING-1/SEC STEAL-044%
16:39:36 Q0-00002(00000)                              DORMANT-00017
16:39:36 Q1-00000(00000)            E1-00000(00000)
16:39:36 Q2-00000(00000) EXPAN-001 E2-00000(00000)
16:39:36 Q3-00002(00000) EXPAN-001 E3-00000(00000)
16:39:36
16:39:36 PROC 0000-001% CP        PROC 0001-001% CP
16:39:36 PROC 0002-002% CP        PROC 0003-001% CP
16:39:36
16:39:36 LIMITED-00000
```

Figure 3-20 VM command - cp indicate

cp query srm

The **cp query srm** command displays some of the current z/VM system tuning parameters. Each of these parameters has a default setting that is appropriate for most environments. Later chapters in this book discuss potential adjustments to the defaults. The System Resource Manager (srm) parameters contained in this display include:

- iabias - interactive bias, used to favor short transactions
- ldubuf - controls z/VM's tolerance for guests that induce paging
- storbuf - can be used to encourage z/VM to overcommit main memory
- dispatching minor timeslice - controls the length of time that a single user holds onto a processor

```
16:41:24 Q SRM
16:41:24 IABIAS : INTENSITY=90%; DURATION=2
16:41:24 LDUBUF : Q1=300% Q2=200% Q3=150%
16:41:24 STORBUF: Q1=800% Q2=700% Q3=600%
16:41:24 DSPBUF : Q1=32767 Q2=32767 Q3=32767
16:41:24 DISPATCHING MINOR TIMESLICE = 5 MS
16:41:24 MAXWSS : LIMIT=9999%
16:41:24 ..... : PAGES=999999
16:41:24 XSTORE : 0%
```

Figure 3-21 VM command - cp query srm

cp indicate queues

The **cp indicate queues** command displays user IDs and their associated dispatching queue (Figure 3-22).

```
16:37:11 INDICATE QUEUES
16:37:11 OPERATOR       Q1 R00 00000000/00000001 LNXIHS        Q1 PS  00087663/00
087564
16:37:11 LNXDB2         Q0 PS  00118266/00118252 LNXSU4        Q2 PS  00033352/00
033110
16:37:11 LNXDB2   MP01 Q0 PS  00000000/00000000 LNXSU1        Q3 PS  00086785/00
103755
16:37:11 LNXWAS         Q3 PS  00682417/00786432 TCPIP         Q0 PS  00000718/00
000311
```

Figure 3-22 VM command - indicate queues

cp query alloc page

The **cp query alloc page** command can be used to determine how much paging space is used by the system.

3.8 Velocity monitoring tools

Another monitoring tool in use today is ESALPS, the Linux Performance Suite of products provided by Velocity Software. The products that make up this suite of tools include:

- ESAMAP

  The VM Monitor Analysis Program, providing performance reports on all aspects of VM/ESA and z/VM performance.

- ESAMON

  The VM Real Time Monitor, providing real-time analysis of performance.

- ESATCP

  The network and Linux data collection program.

- ESAWEB

  A very fast VM-based Web server.

In addition to the four products, ESALPS provides a Web-based interface to view performance data through a Web browser and many control facilities.

3.8.1 Monitoring requirements

There are many requirements for data collection met by ESALPS. Data is provided for:

- Capacity planning

  Long-term data in the form of a performance database (PDB™) is needed as input to long-term capacity planning and trend analysis. Full historical data functions are provided with collection and many forms of data extraction tools.

- Performance analysis

  Trend data enables an analyst to detect performance changes in any of thousands of potential problem areas. The performance database allows analysts to determine what of many potential changes occurred in the system. Reporting on specific periods of time can be performed, enabling an in-depth performance analysis of performance problems.

- Real-time performance

  Beyond the traditional *entry level* real-time performance reporting of the top users and system utilization, real-time performance analysis is provided for all subsystems, user activity, and Linux (and many other platforms) servers. Network data is also provided real time.

- Linux data

  With the advent of virtual Linux server farms on z/VM, performance data is required.

3.8.2 Standard interfaces

ESALPS uses standard interfaces for all data collection. The advantage to using the standard interfaces provided is that when there are a multitude of releases and distributions available, the standard interfaces provide consistent data sources. Supported interfaces include:

- z/VM provides a *monitor interface* that has been available since 1988. Since then, this interface has provided a consistent view of performance of VM systems.
- Network performance is collected using simple network management protocol (SNMP), the standard for network management.
- NETSNMP, an open source software package, provides host data for Linux and other platforms.
- VM application data interface is used by applications to insert data into the monitor stream consistent with the standard monitor interface. ESATCP uses this interface to ensure consistent data collection that allows full integration of Linux and VM data.

3.8.3 Performance database

ESALPS provides both real-time data and historical data for in-depth analysis. The performance data is collected daily with a one-minute granularity based on the monitor interval. A longer term archive is collected, usually with a granularity of 15 minutes. This performance database (PDB) includes VM data, Linux data, and network data.

3.8.4 Real-time monitoring with ESAMON

ESAMON uses:

- Historical reporting
- Linux reporting
- Network reporting

Velocity Software has been in business since 1988 supporting the VM environment. With a focus on VM, and now z/VM, Velocity Software added TCP/IP network analysis and then Linux, Microsoft® Windows® NT, Sun™, and other platforms to the product data collection facilities.

3.9 Other monitoring tools

Other performance monitoring software products are available that may provide similar system statistics as the data that is referenced in this book. Some of those products include:

- BMC Mainview monitor for z/VM and Linux
- Computer Associates MICS

This is not a complete list of all products available on the market.

3.10 Capacity planning software

Successful capacity planning involves projecting future needs based on past trends and business plans for growth. Although hardware costs continue to decline, software and personnel costs are rising. Since software and personnel costs are tied to hardware capacity, capacity planning is still important.

Capacity planners judge when to buy more hardware based on current throughput and performance, plus growth expectations for the IT workload. Performance is then a key indicator when trying to predict the timing for a processor upgrade. High processor utilization is desirable from a cost viewpoint, but only so long as performance does not suffer.

Capacity planners need tools more sophisticated than rules-of-thumb. Good modeling software enables the planner and her IT department to plan hardware upgrades before poor system performance becomes a business growth inhibitor. Modeling tools that are available from IBM to assist IT departments with effective capacity planning for System z environments include the z/VM Capacity Planner and the Tivoli Performance Modeler. A System z processor that has only z/VM and Linux images can be modeled by the IBM z/VM Planner for Linux Guests on IBM System z Processors (IBM employees and authorized IBM Business Partners can use the tool with their customers). A System z process that is running a combination of z/VM, Linux, and z/OS images can be modeled using a combination of the IBM z/VM Planner for Linux Guests on IBM System z Processors and Tivoli Performance Modeler.

IBM z/VM Planner for Linux Guests on IBM System z Processors

The z/VM Planner for Linux Guests on IBM System z Processors (z/VM-Planner) is a PC-based productivity tool that runs under MicroSoft™ Windows™. It is designed to provide capacity planning insight for IBM mainframe processors running various Linux workload environments as guests under VM. Input consists primarily of z/VM guest definitions and capacity requirements for each intended Linux guest. The total guest capacity requirement is calculated based on the guest definitions and, since it is probable that all guests do not have peak utilization at the same time, a user-specified complementary peak percentage is used to calculate the true combined capacity requirements of the z/VM guests being sized. The tool also models z/VM processor requirements when merging new guests into existing VM images. The resulting guest capacity requirement is combined with that of VM to support the entire complement of guests.

The z/VM-Planner produces several graphs that depict the modeled system. The graphs illustrate the size of processor needed to support the modeled workload, the percent of that processor used by each application in the workload, and the estimated minimum and maximum processing capacity needed for the workload. Figure 3-23 represents the minimum and maximum capacity needs for a modeled workload with five Linux guests running a mixture of Web applications, databases, and file servers.

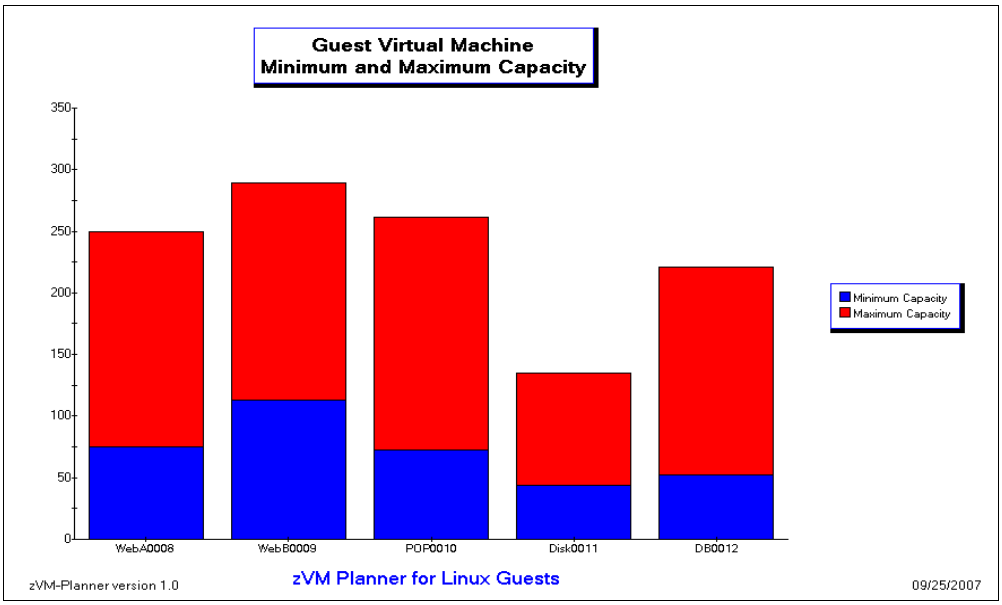


Figure 3-23 z/VM-Planner graph of modeled system

Please contact your IBM representative if you are interested in learning more about this tool.

IBM Tivoli Performance Modeler

IBM Tivoli® Performance Modeler for z/OS® V2.3 is a PC-based performance modeling and capacity planning tool that runs on MicroSoft™ Windows™. It can be as portable as your PC or mobile computer. IBM Tivoli Performance Modeler for z/OS V2.3 is designed for system management on the IBM eServer™® zSeries® and S/390®®. With growing operating system complexity and the huge impact of responding to workload changes, basic Central Processor Unit (CPU) utilization is generally no longer sufficient information for performing capacity planning. IBM Tivoli Performance Modeler for z/OS V2.3 can be used to model the impact of changing:

- Number and speed of CPUs
- Disk I/O response times
- Paging rates (auxiliary and expanded memory paging)
- Logical partition (LPAR) definitions and parameter changes

The Tivoli Performance Modeler takes input from z/OS images, including system and application performance data, as a basis for the modeling program. The user can then make changes to the type of processor and modify the workload configuration to create a new model. The modeler then creates a series of graphs to depict the modeled workload. Figure 3-24 shows a complex workload of batch, database, and online applications.

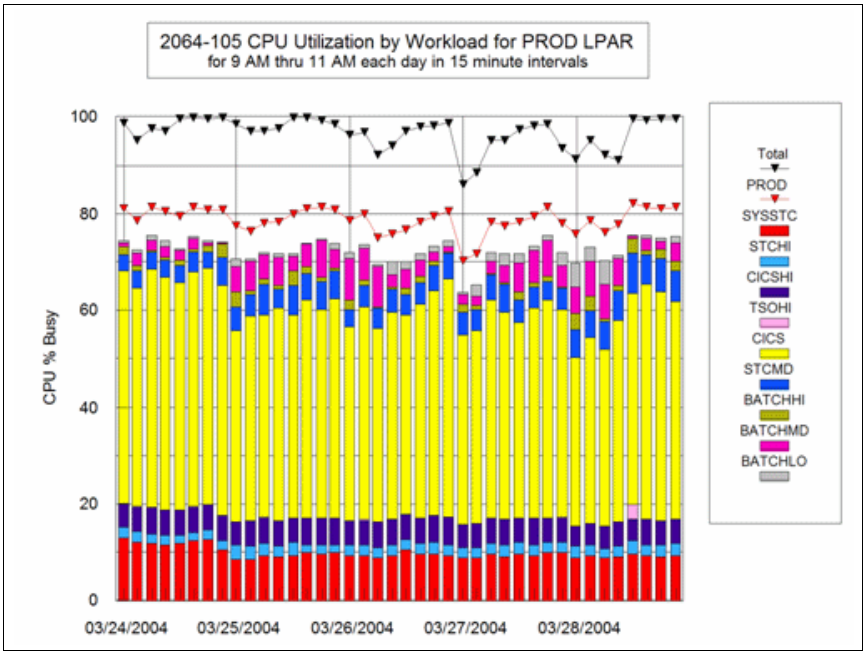


Figure 3-24 Tivoli Performance Modeler graph of modeled systems

4

z/VM storage concepts

This chapter introduces z/VM storage-related concepts. Topics include:

- The z/VM storage hierarchy
- Guidelines for allocation of z/VM storage
- z/VM use of storage
- Virtual storage as seen by Linux guests
- Influencing z/VM storage management
- Paging and spooling

© Copyright IBM Corp. 2003, 2008. All rights reserved.

4.1 The z/VM storage hierarchy

Both z/VM and Linux use what is known as virtual storage, or virtual memory. Usually, the term *storage* is used by z/VM, while *memory* is used by Linux. In this chapter, storage is generally used to explain z/VM-related concepts, while in other chapters related to Linux, memory is used. Keep in mind that these different terms refer to the same thing.

Briefly, virtual storage is a method of allowing more programs and data to share real (physical) storage at the same time. At any given moment, a program can only access a very small amount of storage, even over a period of time. It is very unlikely that it will access more than a fraction of the total storage that it has been assigned to use. Virtual storage systems use a mechanism called paging that tries to ensure that storage that is actively being used by a program is in real storage; and that another part that is not being actively used is temporarily saved to expanded storage or disks. Then real storage can be available for many other programs to use. In both z/VM and Linux, storage (called memory in Linux) is managed in 4 K pages.

The z/VM storage hierarchy uses three types of storage:

- Main storage

  This storage is directly addressable and fast-accessible by user programs. Both data and programs must be loaded into main storage (from input devices) before they can be processed by the CPU. The maximum size of main storage is restricted by the amount of physical storage.

- Expanded storage

  Expanded storage also exists in physical storage, but is addressable only as entire pages. Physical storage allocated as expanded storage reduces the amount for main storage. Expanded storage is optional, and its size is configurable.

  Expanded storage acts as a fast paging device. As demand for main storage increases, z/VM can page to expanded storage. Because it exists in physical storage, paging to expanded storage can be much faster than paging to Direct Access Storage Device (DASD).

- Paging space

  Paging space resides on DASD. When paging demands exceed the capacity of expanded storage, z/VM uses paging space.

The relationship between the types of z/VM storage is depicted in Figure 4-1.

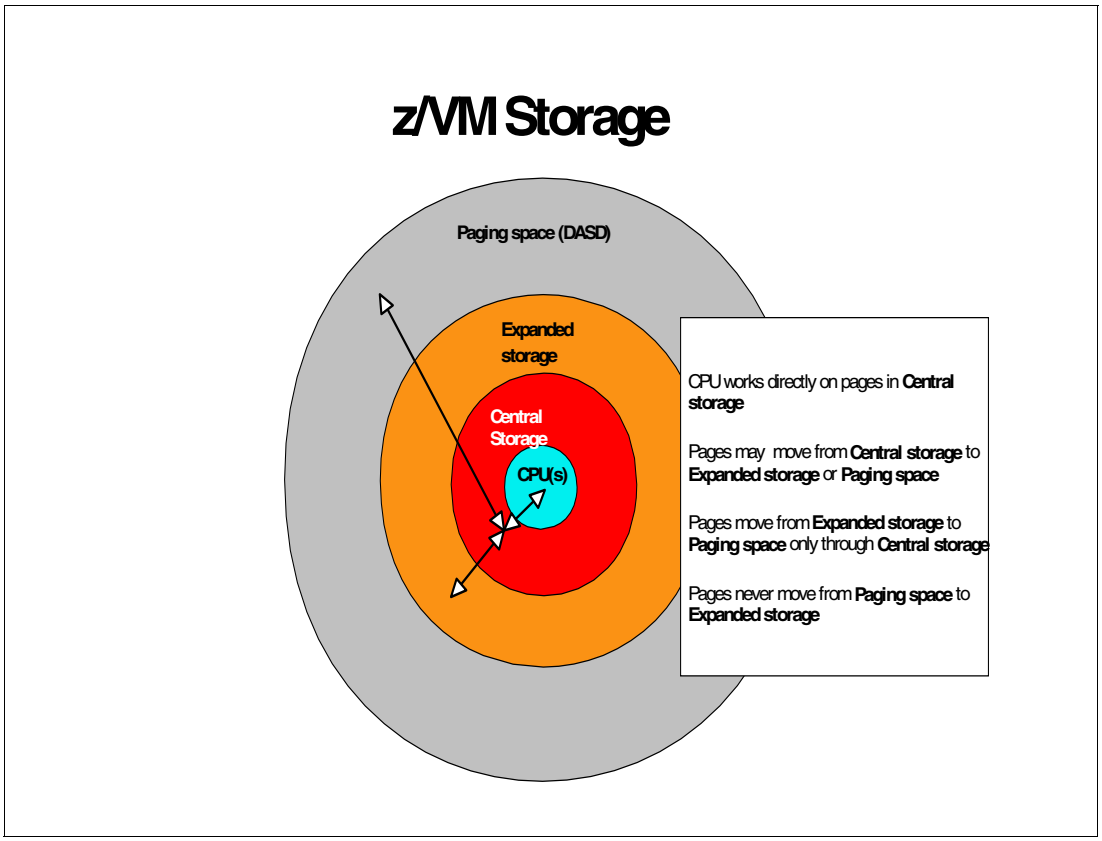


Figure 4-1 The relationship between the types of z/VM storage

The combination of main storage, expanded storage, and paging spaces forms the z/VM virtual storage address space. As illustrated in Figure 4-1, pages move within z/VM virtual storage to accommodate demands:

- z/VM guests run in main storage.

  When dispatched, guests execute in main storage. Not all guest pages need to reside in main storage when running. Inactive pages can reside in expanded storage or in paging spaces, or in both.

- Paging occurs between main storage and expanded storage.

  As demand for main storage increases, z/VM might move inactive guest pages to expanded storage. When those pages become active, z/VM moves them back to main storage.

- Paging also occurs between main storage and paging space.

  If no expanded storage is configured, z/VM pages between main storage and paging space. If the paging demand exceeds the expanded storage capacity, z/VM also pages to and from main storage and paging space.

- Pages do not move directly between expanded storage and paging space.

  Pages that move from expanded storage to paging space must first be brought to main storage.

4.2 Guidelines for allocation of z/VM storage

As discussed in 1.2.1, “Overcommitting resources” on page 5, overcommitted storage is a normal and desirable situation on z/VM. Storage overcommitment allows z/VM to provide more total resource utilization (and therefore, a lower overall cost). The z/VM storage hierarchy is designed to optimize paging on an overcommitted system.

With 64-bit support, the question arises of whether there is a need for expanded storage (why not configure all physical memory as main memory instead). The general recommendation is still to configure z/VM with expanded storage.

Expanded storage often results in more consistent or better response time. Factors that suggest expanded storage improves response time include:

- Paging will likely occur in a z/VM system.

  The logic for allocating all physical storage as main storage is that paging only occurs if there is a shortage of main storage, and therefore expanded storage only increases this possibility. However, overcommitment of storage on z/VM is a *normal* and *healthy* practice. Therefore, it is better to prepare for paging than to attempt to prevent it.

- z/VM paging algorithms are tuned for expanded storage.

  The paging algorithms in VM evolved around having a hierarchy of paging devices. Expanded storage is the high-speed paging device and DASD is the slower one. Moving pages from main storage to expanded storage is much more efficient than moving pages from main storage to paging DASDs.

- Two GB limitation.

  Before z/VM 3.1.0, VM only supports 31-bit, which means that all of the programs and data have to reside in the 2 GB address space. But even with 64-bit support started from release 3.1.0, parts of CP must still reside below the 2 GB line until the release of 5.3. This can create contention for storage below 2 GB, which can be indicated by significant paging to DASD or expanded storage, and a large number of pages are still available above 2 GB in main storage. The storage contention definitely degrades performance. Systems with a 2 GB constraint may benefit from having more physical storage configured as expanded storage.

  Note: Storage contention can be seen by using the QUERY FRAMES command or from z/VM Performance Toolkit or OMEGAMON.

  As a general estimate, it is good to start with expanded storage configured to be 25% of the physical storage allocated to z/VM. Many systems do not need more than 2 GB of expanded storage regardless of the total storage available. If your system has a 2 GB storage constraint, configure all the storage that cannot be effectively used in main storage as expanded storage. From the result of QUERY FRAMES or Performance Toolkit, the number of frames (> 2 G pages) on the available list can show how much storage can be configured as expanded storage. For more tips about storage configuration, refer to:

  http://www.vm.ibm.com/perf/tips/storconf.html

4.3 z/VM use of storage

The initial 64-bit support is first provided by z/VM 3.1.0, and is incrementally improved through z/VM 5.3. The following discussion describes the use of storage below 2GB-line throughout these releases, so as to tune your VM for better performance accordingly.

While z/VM has 64-bit support and can support programs and data saved above 2 GB of main storage (real storage), there are some areas of the control program (CP) that are limited to 31-bit (2 GB) addressing. From VM 3.1.0 through 5.1.0, all CP code and data structures have to reside below the 2 GB line, and most of the CP code continues to use 31-bit addressing. A guest page can reside above 2 GB. It could continue to reside above 2 GB and have instructions in it executed or data referenced without being moved. But, if a page is referenced by CP or requires CP processing, it must be moved to a frame below the 2 GB line in main storage before it can be manipulated by the 31-bit CP code. This includes things such as I/O channel programs and data (both SSCH and QDIO), simulation of instructions, and locked pages (like QDIO structures for real devices).

For example, if a guest is executing an I/O while the channel programs and data reside in a guest page above 2 GB, CP needs to do page translation to process the virtual I/O. During the translation, the guest page is moved below 2 GB in main storage and locked until the I/O completes. The page remains in the main storage until it is stolen or released. This may cause storage contention below the 2 GB line. When that occurs, pages have to be moved out of main storage and brought back again in a short period of time.

Note: A page can be stolen during the CP stealing process, which is started when the number of available pages below 2 GB is too low. During the CP stealing process, pages below 2 GB are moved to expanded storage or paging DASDs. Since expanded storage is much faster than paging DASDs, it is better to configure expanded storage in the system.

A page in main storage can be released explicitly or when the guest logs off or goes through system reset.

Usually, a system that is constrained by 2 GB has symptoms of high paging activities and a large amount of space in main storage above 2 GB. This can be determined by using CP INDICATE LOAD or CP QUERY FRAMES commands. The output of these commands may vary from release to release. For details about the commands and related output, refer to the *z/VM CP Commands and Utilities Reference*.

The reason why all CP programs and data have to reside below 2 GB is because of CP's address space. CP runs its own address space called system execution space (SXS), which can only be up to 2 G in size. Prior to z/VM 5.2, the SXS was identity-mapped, that is, all the logical addresses are the same as real addresses in main storage. Since z/VM 5.2, the SXS is no longer identity mapped, which allows pages in the SXS to reside in any part of storage, including the frames above 2 GB line. When CP wants to reference a guest page, it can map the page to a logical page in SXS and there is no need to move it below 2 GB. In addition, the CP control block that maps real storage was moved above 2 GB, and some additional CP modules are changed to 64-bit addressing. These changes eliminate most of the limitations to move pages below the 2 GB line, except the page management blocks (PGMBKs) in z/VM 5.2.

PGMBKs in z/VM 5.2 must still reside below 2 GB in real storage. A PGMBK includes the page tables and related page management tables for a 1-megabyte segment of virtual storage. Each PGMBK is 8 K in size and pageable. When it resides in storage, a PGMBK is backed by two contiguous frames below 2 GB in main storage. If at least one page of the 1-megabyte segment of virtual storage is backed by one of the frames in real storage, the

PGMBK must reside in main storage. This limitation sets a maximum of 256 GB for the amount of active virtual storage that z/VM 5.2 can support. It may become an affecting factor when systems are getting larger. PGMBK usage can be checked by using OMEGAMON or z/VM Performance Toolkit. For more details refer to *z/VM Performance Toolkit Guide*, SC24-6156.

The limitation on PGMBKs is removed now in z/VM 5.3. PGMBKs can reside above the 2 GB line in real storage. This greatly increases the maximum amount of in-use virtual storage supported by VM. With z/VM 5.2, the PGMBKs must reside below 2 GB. If the entire first 2 GB of real storage could be allocated to resident PGMBKs, the limit of in-use virtual storage is 256 GB. While with z/VM 5.3, PGMBKs can reside in any place of main storage, the limit of in-use virtual storage is determined by the size of total real storage of the z/VM system. The maximum size of real storage supported by z/VM is now increased from 128 GB in release 5.2 to 256 GB in release 5.3. A system with 256 GB real storage can support at most 21.3 TB of in-use virtual storage. That is greatly beyond the CP storage management design limit of 8 TB. In addition, the management of contiguous frames is also further improved, and the search for single and contiguous frames on the available list is much more efficient in z/VM 5.3. For more details about performance of z/VM 5.3, refer to:

http://www.vm.ibm.com/perf/reports/zvm/html/zvm530.html

4.4 Virtual storage as seen by Linux guests

Virtual storage refers to the combination of main storage, expanded storage, and paging space. Linux guests running under z/VM see storage as a contiguous area extending from a low address of zero to a high address equal to the virtual machine size. However, these pages might not be contiguous in z/VM virtual storage, and might be moved out of main storage to accommodate demand. In fact, z/VM might move a Linux guest's pages out of main storage without notifying the guest.

4.4.1 The double paging effect

The z/VM storage management process of paging Linux a guest's pages out of main storage without notifying the guest may lead to a situation referred to as *double paging*, an effect illustrated in Figure 4-2.

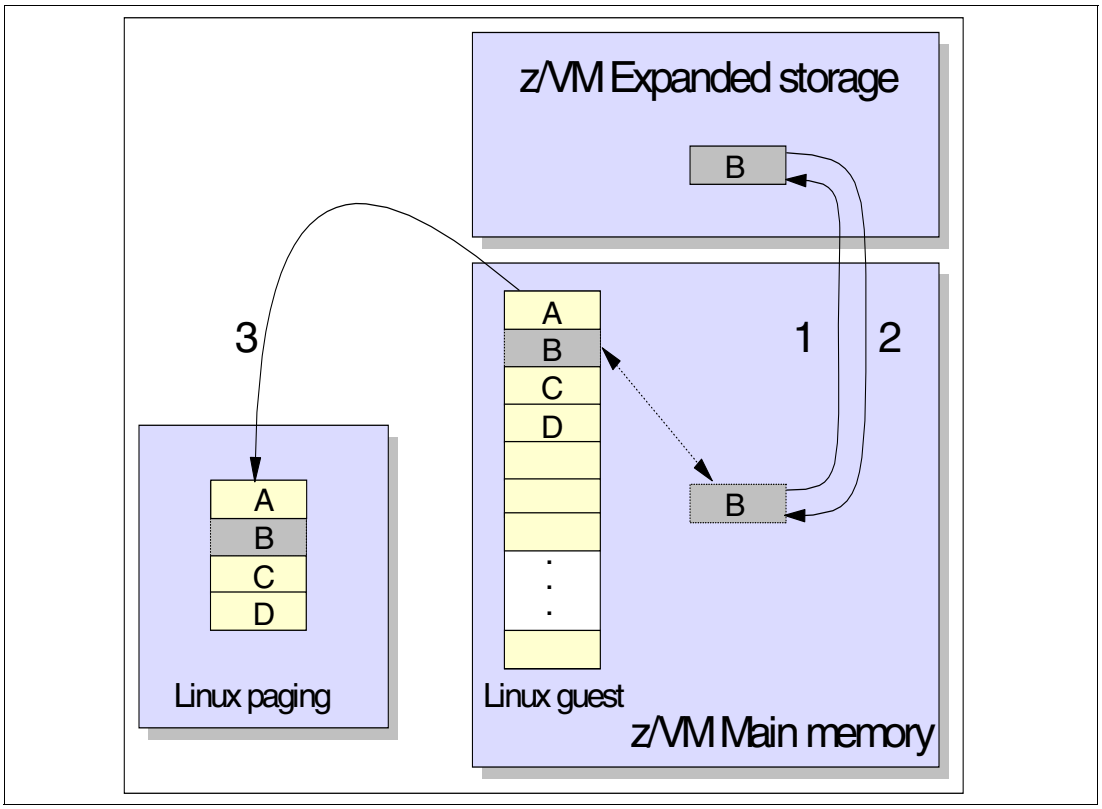


Figure 4-2 Illustrating the double paging effect

Figure 4-2 depicts storage pages used by a Linux guest. The event sequence leading to double paging is denoted by the numbered arrows:

1. Page B is moved out from z/VM main storage.

   This page is mapped to a running Linux guest. After a period of inactivity, z/VM moves the page to expanded storage.

   Note: In this example, we show the page-out occurring to expanded storage. If no expanded storage was configured, the page-out would occur to paging space, an even more expensive operation.

   The page is now available for demand page-in. z/VM has stolen the page, but the Linux guest is unaware that the page-out occurred.

2. Page B is paged back into z/VM main storage in response to a page-out attempt by the Linux guest.

   To alleviate a storage constraint, Linux identifies inactive pages (including page B) to move to its swap device. The Linux guest page faults when attempting to access page B. This, in turn, causes z/VM to page-in storage page B to satisfy that request.

3. Linux completes its page-out attempt by moving page B to its swap device.

After page B is paged back into main storage, Linux pages it out to its swap device. In the end, z/VM has moved pages back into main storage simply to allow Linux to move it out again.

Double page faults are not unique to Linux. z/OS running under z/VM can experience the same effect. It is a result of two parties attempting to manage memory at the same time. The solution is to ensure that one party does not attempt to page:

- Make the Linux guest virtual machine size small enough for z/VM to keep in main storage.

  Use working set size estimation techniques to determine how much storage should be given to a Linux guest. It is better to give slightly more storage to the Linux guest than the working set size. Double paging might still occur if many smaller Linux guests compete for z/VM main storage.

- Make the virtual machine size large enough that Linux does not attempt to swap.

  This can lead z/VM to frequently page fault. This can be improved by PAGEX/PFAULT, an asynchronous mechanism that allows z/VM to inform Linux that a requested storage page is not resident in z/VM main storage. On receiving this notification, Linux attempts to dispatch another process.

- Cooperative Memory Management (CMM).

  z/VM and Linux attempt to manage storage separately, which leads to double paging. To avoid the problem, a new solution is needed to make z/VM and Linux cooperate more efficiently in storage management. Storage usage information is passed from Linux guests to z/VM. Then the z/VM steals pages based on the current usage information. If the Linux guest accesses pages removed by z/VM, discard faults are delivered, and the Linux guest recreates the content of the discarded page. More details can be found in 6.5, “Cooperative memory management (CMM)” on page 75.

- Collaborative Memory Management Assist (CMMA).

  CMMA is the latest solution to solve the double paging issue. The pages in use are flagged when allocated and so later it is easy to distinguish if they are able to be reused. Contrary to the CMM solution above, the collaborative memory management assist is completely controlled by the Linux guest. More details are in 6.6, “Collaborative memory management assist (CMMA)” on page 76.

4.4.2 Allocating storage to z/VM guests

Storage of z/VM virtual machines is assigned in the user directory entry. Two values in the directory entry specify storage for each user, as shown in Example 4-1:

- The first value is the storage size of the guest’s virtual machine when it logs on.
- The second value is the maximum storage size to which the virtual machine can change itself after it has logged on.

Example 4-1 A user directory entry for Linux LXNMR1

```
USER LNXMR1 LNX4ME 512M 2G G
   INCLUDE IBMDFLT
   IPL CMS
   MACHINE XA
   OPTION LNKNOPAS APPLMON
   NICDEF C200 TYPE QDIO LAN SYSTEM VSWITCH1
   MDISK 0191 3390 141 20 LX6U1R MR
   MDISK 0101 3390 1001 1000 LXCD3B
   MDISK 0201 3390 0001 3338 LXCD3A
```

The initial and maximum storage sizes are specified in the user directory entry, and the virtual storage size can be changed by commands. If a guest changes its virtual machine size, the storage is reset, and the initial program load (IPL) occurs again. As a result, changing a virtual machine size is something usually done by a human user, not by an operating system running in a virtual machine. Table 4-1 lists some of the storage allocation activities.

Table 4-1 Storage allocation activities

Activity	Command or directory entry
Initial storage allocation	USER directory entry
Maximum storage allowed	USER directory entry
Change storage allocation	DEFINE STORAGE command
Modify initial storage settings	DIRMAINT command[a]
Display storage allocation	QUERY VIRTUAL command

a. The Directory Maintenance Facility (DirMaint™) must be enabled in order to use DIRMAINT commands.

4.4.3 VDISKs

VDISKs are virtual disks, emulated disks that z/VM creates in virtual storage. VDISK is backed by a storage address space instead of by DASDs. When used, VDISK resides in main storage, which make it very fast. Typical VDISK usage for Linux guests is as a fast swap device (see 7.4.3, “The advantages of a VDISK swap device” on page 89).

For VDISK I/O, z/VM does not move the Linux guest’s swap file I/O buffers below the 2 GB line in main storage. However, the page and segment tables that define the VDISK address space (PGMBKs) are not pagable and must reside below the 2 GB address in main storage before z/VM 5.3. These PGMBKs can increase contention for storage below 2 GB.

If your system has plenty of real storage, latency in Linux swap I/O can be reduced by using VDISK as a swap device. If the system is storage-constrained below 2 GB but rich above, VDISK might be appropriate for your system if it is not defined as unnecessarily large.

4.4.4 Minidisk cache (MDC)

Minidisks cache (MDC) is a data-in-storage technique that attempts to improve performance by decreasing the I/O to DASD required for minidisk I/O. Minidisk cache decreased DASD I/O at the expense of real or expanded storage. When paging to DASD increases, the amount of real and expanded storage decreases for the use of MDC. This may increase paging I/O in z/VM. However, paging in z/VM is normal and healthy. If using MDC greatly decreases the real DASD I/O with some increase in paging I/O, it is still worth using.

All minidisks that meet the requirements for minidisk cache are enabled by default. The CP arbiter function determines how much real or expanded storage can be used as minidisk cache and paging. If the arbiter is favoring paging or minidisk cache more than is optimal for the system, it can be biased for or against minidisk cache by using commands SET MDCACHE STORAGE BIAS or SET MDCACHE XSTORE BIAS.

The z/VM minidisk cache is implemented with a system cache where the storage for the cache is shared with all the users on the system. To prevent one of the users from negatively impacting other users on the same system, a fair share limit is enforced by CP. If a user reaches the fair share limit, the I/O completes without inserting the data into the cache. The

fair share limit can be overrided by using the NOMDCFS operand on the OPTION statement in a user directory entry. We recommend that he key users, such as server machines or guests that do I/O on behalf of many users, turn off the fair share limit.

Since the use of minidisk cache usually leads to an increase in paging, the paging configuration should be appropriate. We recommend that the block size of page reads should not go below 10, and paging space should not be mixed with other types of data, including spool space. Balancing the paging I/O over multiple control units is better.

4.5 Influencing z/VM storage management

Several factors can influence how z/VM manages storage. These include:

- Update system software to the current level.

  Several improvements in z/VM storage management can be found release by release. Utilize the latest technology with higher performance. More details about the storage management improvement at different releases can be found in the z/VM performance report at:

  http://www.vm.ibm.com/perf/reports/zvm/html/index.html

- Ensure that there are enough paging DASDs and that the paths to them are not very busy.

  This makes paging as fast as possible. Use multiple devices over as many DASD control units as it can to permit overlapped I/O. Use entire dedicated volumes for z/VM paging, instead of fractional volumes, that is, do not mix page spaces with other space (user, spool, tdisk, and so on), and make sure to define enough space so that you can do block paging. Try to provide at least twice as much DASD paging space as the sum of the Linux guests' virtual storage sizes.

- Use expanded storage for paging.

  As discussed in 4.2, "Guidelines for allocation of z/VM storage" on page 36, expanded storage can improve overall performance. A good starting point is to define 25% of the real storage as expanded storage. If your system is not constained by 2 GB line, 2 G of expanded storage is usually enough.

- Use shared segments.

  Typically, the Conversational Monitor System (CMS) is run from a named saved system (NSS), but other operating systems can be, too. The advantage is that only one copy of the operating system resides in storage accessible to all virtual machines. We discuss the creation of a Linux NSS in 6.3, "Exploiting the shared kernel" on page 65. By using a NSS, only one copy of the Linux kernel is saved in main storage, and the pages making up this kernel in main storage can be shared with many virtual machines. This decreases storage requirements, since there is not a separate kernel for each guest.

- Explore discontiguous saved segments (DCSS) when appropriate.

  One of the unique advantages of z/VM is the ability to share segments amongst virtual machines. There are a number of segments that may be predefined on your system. If they are not used, they should not be loaded into main storage. Use the QUERY NSS MAP command to determine how many users are using a segment. When a segment is shared by several virtual machines, there is only one set of PGMBKs in the main storage, which can greatly reduce the storage requirement if the content in the segment is required by a large number of Linux guests.

- Adjust SRM controls.

  These control how z/VM manages virtual storage. System Resource Management (SRM) parameters are discussed in 8.4.1, “Global System Resource Manager (SRM) controls” on page 114.

- Ensure that system operation information records are properly retrieved and stored.

  z/VM allows for the collection of various accounting, error, and symptom records during operation. This data is stored in main storage until retrieved by designated virtual machines. A new installation of z/VM typically defines virtual machines that are automatically logged on at IPL:

  – The DISKACNT virtual machine retrieves accounting records.
  – The EREP virtual machine retrieves error records.
  – The OPERSYMP virtual machine retrieves symptom records.

  If these virtual machines stop retrieving the data, available system storage can be greatly reduced. Further details about these information collection facilities can be found in *z/VM V5R3.0* z*/VM: System Operation*, SC24-6121.

- Use the CP SET RESERVED command to reserve pages for an important guest.

  The CP SET RESERVED command reserves real storage pages to a virtual machine. Although it can be used to improve performance of specified guests, it should be used sparingly. Otherwise, it would diminish the performance of the entire system.

Note: The Virtual=Fix (V=F) and Virtual=Real (V=R) types of virtual machines, also known as preferred virtual machines, which allow guest pages to reside permanently in real storage, are no longer supported starting from z/VM 5.1.0.

For z/VM systems before 5.1.0, V=F or V=R are still optional. The V=F option is only available if the hardware processor is IPLed in basic ESA/390 mode and not in LPAR mode. There are limitations and restrictions associated with V=R and V=F. Both can adversely affect overall system performance.

Although these factors influence virtual storage management, not all of them can be considered beneficial. In general, options that enable resource sharing are recommended for tuning overall system performance.

4.6 Paging and spooling

z/VM utilizes specially allocated DASD space for system storage that is available to all virtual machines running within that system. This DASD space is owned and controlled by the system (CP) and has distinct uses.

As already discussed, paging DASD space is utilized for virtual machine paging along with main storage and expanded storage. Paging needs to be as fast as possible. Because the data in paging space is temporary, all the paging data is discarded when z/VM is shut down.

Here are some guidelines to set up the z/VM system for paging:

- Provide at least twice as much DASD paging space as the sum of the Linux guests’ virtual storage size minus the main memory size. The CP QUERY ALLOC PAGE command can be used to see how much paging space the system has.
- Use entire volumes for z/VM paging space. In other words, never mix paging I/O and non-paging I/O (user space, spooling, tdisk, and so on) on the same pack.

- Try to distribute the paging volumes over as many DASD control units as you can, and implement a one-to-one relationship between paging CHPIDs and paging volumes. This makes the I/O concurrently occur to several paging DASDs through multiple paths.
- If the paging control units support NVS or DASDFW, turn them on by using the CP commands SET NVS ON or SET DASDFW ON.

Spool space is used to store data that is common to all virtual machines. Because z/VM simulates an environment of independent operating systems, each of these virtual machines has a designated reader, punch, and printer, just like the computer systems of old. Data sent to a virtual machine resides in the reader until deleted or read into a user's minidisk, much like a mail in-box. Data that is created as output is directed to the virtual printer or to the virtual punch.

Spool space is also used for executable data or systems stored for access of all virtual machines, such as NSS files. This creates a common location for system code, rather than each virtual machine requiring individual storage space. Because the contents of spool space are valid data, they are preserved across z/VM shutdowns and IPLs.

Along with DASD space that is owned by specific virtual machines (better known as minidisks), z/VM allows for temporary minidisk space. This space can be allocated as needed by a virtual machine from the tdisk space created on the system, which allows you to define disks of various sizes with less consideration for placement. This is meant only to hold data that is being processed or that does not need to be retained. When z/VM is shut down, all data in the disk space is discarded. If a temporary minidisk (tdisk) is used, the minidisk cache should be disabled for that minidisk.

5

Linux virtual memory concepts

In this chapter we discuss how Linux uses virtual memory. We present Linux memory management concepts and consider how these concepts affect Linux guests running under z/VM. Topics include:

- Components of the Linux memory model
- Linux memory management
- Aggressive caching within Linux
- Conclusions for sizing z/VM Linux guests

© Copyright IBM Corp. 2003, 2008. All rights reserved.

5.1 Components of the Linux memory model

We begin examining Linux memory by looking at the components of memory allocation.

5.1.1 Linux memory

When Linux boots, one of its first tasks is the division and allocation of memory resources. Memory is divided into three main components:

- Kernel memory

  Kernel memory is where the actual kernel code is loaded, and where memory is allocated for kernel-level operations. Kernel operations include:

  - Scheduling
  - Process management
  - Signaling
  - Device I/O (including both to-disk and to-network devices)
  - Paging and swapping

  Pages to host the kernel code are marked at boot time as reserved and are not available for later allocation. The kernel is loaded at the bottom of the memory. Kernel operations use memory of the buddy or slab objects.

- User memory

  User memory is where all application code is loaded:

  - Anonymous memory
  - Shared memory (System V or POSIX API)
  - Memory mapped files

  The need for user memory is satisfied from the master page pool.

- Page cache memory

  Page cache memory caches pages, as the name implies. The Linux kernels 2.4 and later have no separated page and buffer caches anymore. The page cache is filled when read and write operations occur. It hosts data of various sources:

  - Read and write operations of regular files
  - Memory mapped files
  - Operations against block device files

  Page cache is intended to speed up overall system performance by reducing I/O operations (which are inherently slower than memory access).

The kernel keeps track of all the pages using a page array structure.

Note: Linux uses an aggressive caching policy intended to reduce I/O when allocating main memory, the theory being that memory is better utilized as cache, rather than leaving it unused and available. This policy is illustrated in 5.2.2, “Aggressive caching within Linux” on page 48.

5.1.2 Linux swap space

Linux typically asks for the creation of a swap device during initial installation of a system. For periods of high memory demand, Linux temporarily moves less frequently accessed memory pages to its swap devices. The freed memory pages then become available for other use.

When a memory page residing on a swap device is accessed, Linux has to move it back into an available real memory page. Options for defining swap space for Linux on System z are discussed in Chapter 7, “Linux swapping” on page 79.

Swapping versus paging

Swapping is the process of moving an entire address space to a swap device. *Paging* is the process of moving pages of memory to swap space. Recent kernels do not really swap anymore but write pages of memory to the swap device or swap file. Because of its inherent inefficiency (the old swapping requires frequent, expensive context switches) swapping has been modified. Although Linux now utilizes a paging algorithm, the name swap device has been retained.

Readers may still get confused when looking at a Performance Toolkit report like the one in Example 5-1, where both paging and swapping rates are reported.

Example 5-1 Linux memory and activity details panel

```
Linux Memory Util. & Activity Details for System LINUX001

   Total memory size       493MB      Swap space size          140MB
   Total memory used       298MB      % Swap space used       0.06%
   Used for buffer         102MB      Swap-in rate               0/s
   Used for shared           0MB      Swap-out rate              0/s
   Used for cache          225MB      Page-in rate               0/s
   Total free memory        40MB      Page-out rate        25.866/s
```

These are values as available in the Linux guest by Linux commands like **procinfo**. The values mean:

- Page in: the number of disk blocks paged into memory from disk. This means disk I/O in general.
- Page out: the number of disk blocks paged from memory to disk. That means disk I/O in general.
- Swap-in rate: the number of memory pages paged in from swapspace.
- Swap-out rate: the number of memory pages paged out to swapspace.

The activity reported for page-in and page-out rate could be any disk I/O, like the flushing of cached file system pages, but it is not related to writing to the swap space. More information about disk I/O is included in Chapter 10, “Tuning disk performance for z/VM Linux guests” on page 145.

5.2 Linux memory management

Linux memory is divided into *pages*, and each page is usually 4096 bytes in size for s390 and s390x architecture. Linux itself supports other page sizes but needs the support of the architecture, too. The pages are described by a unique structure. Pages are aggregated into memory zones. 64-bit Linux on System z allocates in the DMA zone and the normal zone. The 31-bit Linux on System z only takes advantage of the DMA zone. Zones could be grouped into nodes, which is currently not the case on s390 and s390x architecture.

5.2.1 Page allocation

There are various mechanisms available to allocate pages. A low-level page allocation routine provides contiguous pages for in-kernel allocations, kernel modules, data areas, page cache, anonymous user pages, and other user-required memory. The kernel uses a slab cache that provides a more sophisticated mechanism for allocations of objects of the same type, large scale users (block I/O, network, inodes, and so on).

A busy Linux system following the cache strategy uses most of the available memory for cacheing and processes. Usually, not all of the needed process pages are loaded in the main memory. Page fault handling is the method to bring required pages back into memory. Accessing invalid pages causes a *page translation* check triggered by hardware. Writing to invalid pages causes a *protection exception* check triggered by the hardware. The hardware then provides the correct address using translation-exception identification. The Linux kernel page fault handler checks whether the address is valid in the Virtual Memory Area (VMA), and if so the page is loaded into memory by page-in, swap-in, or copy-on-write. The stack is able to grow automatically if required.

Pages that are not being used by another process and that have not been changed already are seen as available. These pages can be re-allocated without danger of data loss since the information is backed up on a permanent storage device (disk, tape).

5.2.2 Aggressive caching within Linux

Linux manages its memory without regard to the fact that it is running in a virtual machine. The Linux kernel, like most other UNIX® systems, uses an *always full* concept of memory management. It attempts to load as much information (applications, kernel, cache) into its perceived memory resources as possible and caches it there. The target is to reduce the number of disk accesses as much as possible. Even the fastest disk devices are still magnitudes slower than memory access.

To illustrate Linux aggressive caching, we compare the memory usage of two Linux guests, one running in a 64 MB virtual machine, and the other in a 128 MB virtual machine. Each runs the Mstone workload over an eight-minute interval (the Mstone workload is discussed in Appendix C, “Mstone workload generator” on page 205).

The test run has nearly the same working set size not dependent on the available memory. The bigger memory size is not accounted to available (free) memory, but used for buffers and caches.

In Figure 5-1 we examine memory usage for the kernel, the application, the caches, and the buffers over run time with two different memory sizes.

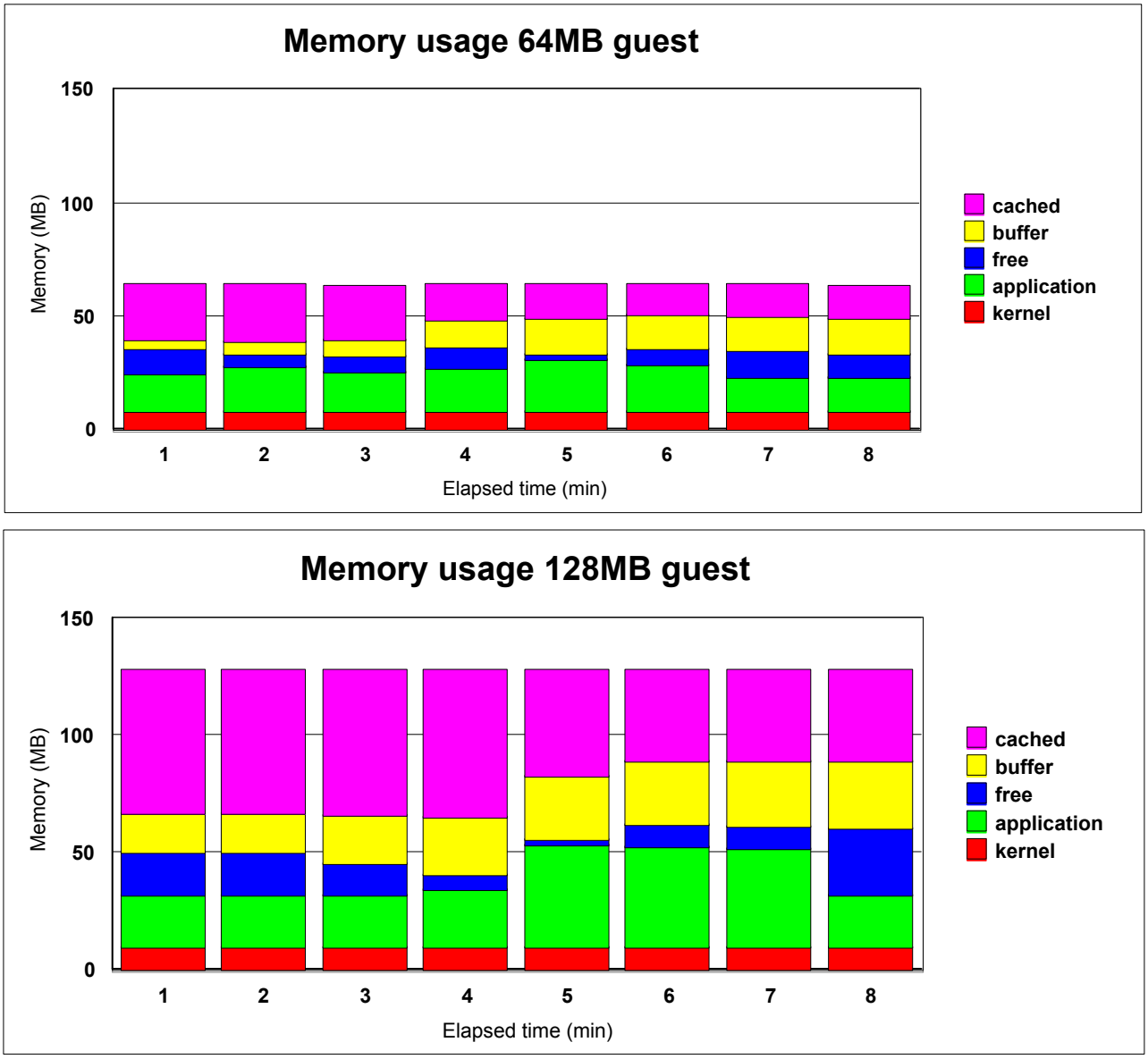


Figure 5-1 Linux memory usage in two Linux guests: 64 MB and 128 MB

Comparing the two Linux guests, we see a similar memory usage pattern: In both cases, additional application memory is obtained at the expense of buffer and cache memory. Reducing the virtual machine size by 50% reduced average caching by 60%.

Note: Although the 64 MB guest required half the amount of memory, no appreciable effect on server response time was noted.

5.2.3 Page replacement

Page replacement has the target of keeping a certain number of pages free for further allocation. The page replacement strategy applies to anonymous pages in memory and

pages in swap cache and page cache. As does other UNIX operating systems, Linux uses the *two-handed clock* mechanism. In an endless loop, a function scans portions of memory to check current page usage:

- The active list is checked for entries that can be added to the inactive list.
- The inactive list is checked for entries that can be reused after content is removed or written back.

Linux memory pages are cleaned up when:

- The kswapd kernel thread runs.

  The kswapd sleeps most of the time. It gets invoked if the number of free pages is less than the pages_low variable. It runs until the pages_high mark is reached or there are no more pages to free.

- pdflush threads are running.

  The purpose of the pdflush daemon is to write back dirty (modified) pages to disk. In older kernels the function has two predecessors, bdflush and kupdated.

  If the number of free pages is below a threshold, pdflush threads are started. These threads write back pages to permanent devices so that the pages can be freed.

  After a specified time, the pdflush threads also wake up to write back pages that are dirty for too long of a time.

- Shrinking in kernel slab caches occurs.

  The pages in slab caches need separate algorithms, but are not so important since slab cache is relatively small.

- The allocation routine initiates direct freeing of pages.

  If the kswapd and the pdflush threads cannot keeping pace with the memory requirements of a process, the allocation routing tries to free some pages.

5.3 Sizing of Linux virtual memory

In a environment with some virtualization layers and many shared resources, having the optimal size of memory is more important, and has more influence on performance, than in a standalone server.

5.3.1 Memory size of the guest and the shared environment

To show the influence of the virtual memory size (and the heap with Java™ applications), we run a series of experiments. In the three-tier environment of the IHS server, WAS server, and DB2 database server, we only changed the size of the virtual Linux guest memory and the heap size of the WAS server. In this experiment we would not drive the Trade6 benchmark to its maximum results except to show that there is an optimal size for an workload. Too few or too many resources can harm the performance of a particular system. Even the overall performance of all the hosted systems could suffer. The transaction throughput is used to gauge how well Trade6 operates in the available Linux memory.

Note: We use an application with Java. The heap size is important, because eventually all memory is touched for removal of objects. The performance is severely degraded if the heap does not fit completely into main memory.

In these experiments we ran the IHS server and the DB2 database server with the same configuration, while changing the WAS server Linux's virtual memory and heap size of WebSphere Application Server. All the measurements were taken after a ramp up and warming up phase with 50 clients.

Table 5-1 depicts the detail configurations of the environment. In our environment, all the servers are configured with only one CPU, IHS server, and DB2 database server, which are configured with a memory of 512 M and 1 G, separately. The WAS server's memory is changed from 1 G to 2 G and 3 G, with a different heap size of the WebSphere Application Server. Note that there is only 4 G memory configured in the entire system, and the memory resource is consumed increasingly in these configurations.

Table 5-1 Details of the configurations

Linux	CPU	Memory	Java heap size
LNXIHS	1	512 M	N/A
LNXDB2	1	1024 M	N/A
LNXWAS	1	1024 M	512 M
	1	1024 M	768 M
	1	2048 M	1200 M
	1	3072 M	2200 M

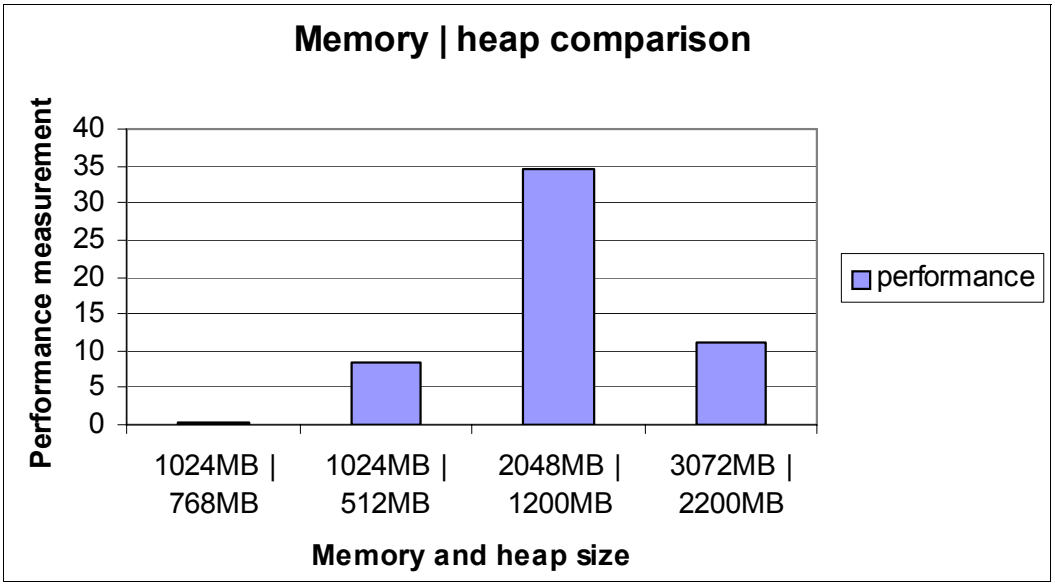


Figure 5-2 Performance of Trade 6 when varying virtual memory and heap size

Figure 5-2 shows the performance result of Trade 6 when varying virtual memory and heap size of the application server. Since these experiments illustrate the effect of virtual memory size on performance, only a moderate workload (50 clients) is driven. We simply use 1/response-time as the performance measurement.

In the first experiment, we ran on 1024 MB of virtual memory, but the heap size of 768 MB was too big to fit into the main memory. Remember that the kernel and libraries also reside in the main memory. Eventually the memory was used completely and the Linux started

swapping. Java frequently used the garbage collection routine, which brings the system nearly to halt in such a situation.

The second experiment uses 1024 MB memory again, but now with a heap size of 512 MB. So no more swapping was seen and the performance was much better.

The third experiment uses 2048 MB of memory and 1200 MB as the heap size. With more resources, it does bring better performance compared to the two previous scenarios.

The fourth experiment brings z/VM in a memory over-commitment by using 3072 MB of memory and 2200 MB as the heap size. The z/VM starts to page moderately to expanded storage. The performance is worse than with 2048 MB memory but slightly better than with 1024 MB of memory. The overhead costs are going up for this workload. Using more users still gives quite good results, as z/VM is used to having an over-commitment ratio and can handle paging well.

In general, performance gets better with more resources. We can see significant differences when we increase both the virtual memory and the heap size from 1024 M to 2048 M from the second to the third scenario. But performance dropped when the virtual memory was increased to 3072 M because of the memory management cost and the over commitment in the last experiment. It shows that more resources do not always bring better performance. The prerequisite is that the virtual memory is set appropriately to the work set and the entire environment. Otherwise, the performance is impacted like in the first experiment. The application server with only 1024 M virtual memory is overloaded by the heap size of 768 M.

5.3.2 Conclusions for sizing z/VM Linux guests

The Linux memory model has profound implications for Linux guests running under z/VM:

- z/VM memory is a shared resource.

  Although aggressive caching reduces the likelihood of disk I/O in favor of memory access, the cost of caching must be considered: Cached pages in a Linux guest *reduce* the number of z/VM pages available to other z/VM guests.

- A large virtual memory address space requires more Linux kernel memory.

  A larger virtual memory address space requires more kernel memory for Linux memory management.

When sizing the memory requirements for a Linux guest, choose the smallest memory footprint that has a minimal effect on performance. Otherwise, all the remaining memory in the long run would be used for buffering and caching.

Note: To determine the smallest memory footprint required, decrease the size of the Linux virtual machine to the point where swapping begins to occur under normal load conditions. At that point, slightly increase the virtual machine size to account for some additional load.

The general rule does not apply to some sort of servers that have special memory needs.

- Database servers

  Database servers maintain buffer pools to prevent excessive I/O to disk. These bufferpools should not be downsized. Otherwise, performance suffers.

- Servers running Java workload (that is, WebSphere Application Server)

  A amount of memory is needed to host the Java heap. The second experiment in hb illustrates how to small heap size degrades the performance even if no swapping occurs.

A good method of keeping shared memory *alive* is to pin pages in memory (that is, do not make them available for replacing or swapping/paging).

To reduce the penalty of occasional swapping that might occur in a smaller virtual machine, use fast swap devices, as discussed in Chapter 7, “Linux swapping” on page 79.

5.4 Observing Linux memory usage

For a quick look at how Linux allocates memory, we use the Linux **free -k** command (the -k option reports memory size in kilobytes). Example 5-2 illustrates the memory usage of a 128 MB virtual memory Linux guest.

Example 5-2 Observing Linux memory usage using the free command

```
free -k
             total       used       free     shared    buffers     cached
Mem:        119360      48036      71324          0       3068      26344
-/+ buffers/cache:      18624     100736
Swap:       719896          0     719896
```

Important points to note in Example 5-2 are:

- The total memory (119360 kB) is less than the total virtual memory size allocated to the Linux guest (128000 kB). The difference (8640 kB) is the size allocated to the kernel.
- The total memory (119360 kB) is equal to the used memory (48036 kB) plus the free memory (71324 kB).
- The used memory (48036 kB) is equal to the buffer (3068 kB) plus the cached memory (26344 kB) plus the used buffers/cache memory (18624 kB).
- The used buffers/cache memory (18624 kB) plus the free buffers/cache memory (100736 kB) is equal to the total memory (119360 kB).
- The free buffers/cache memory (100736 kB) is equal to the free memory (71324 kB) plus the buffer memory (3068 kB) plus the cache memory (26344 kB).

Although there is 100736 kB of *free* memory available, Linux expects applications to use not more than 70% (70515 kB). The operating system expects to use the remainder as buffer/cache memory.

5.4.1 Kernel memory usage at system boot time

To examine kernel memory usage at system boot, we use the **demsg** command, as illustrated in Example 5-3. The **dmesg** command shows all messages issued at boot time.

Example 5-3 dmesg command

```
dmesg | less
.
Dentry cache hash table entries: 32768 (order: 6, 262144 bytes)
Inode-cache hash table entries: 16384 (order: 5, 131072 bytes)
Memory: 116024k/131072k available (4494k kernel code, 0k reserved, 1221k data,
204k init)
.
Mount-cache hash table entries: 256
.
Dquot-cache hash table entries: 512 (order 0, 4096 bytes)
.
IP route cache hash table entries: 2048 (order: 2, 16384 bytes)
.
```

In this example we see that Linux allocates inode and buffer cache at system boot.

5.4.2 Detailed memory usage reported by /proc/meminfo

To examine memory in detail, we query the /proc/meminfo kernel driver, as shown in Example 5-4.

Example 5-4 Detailed analysis of Linux memory using /proc/meminfo

```
# cat /proc/meminfo
MemTotal:       119360 kB
MemFree:         69480 kB
Buffers:          3060 kB
Cached:          26352 kB
SwapCached:          0 kB
Active:          19560 kB
Inactive:        16900 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       119360 kB
LowFree:         69480 kB
SwapTotal:      719896 kB
SwapFree:       719896 kB
Dirty:              56 kB
Writeback:           0 kB
AnonPages:        6920 kB
Mapped:           6592 kB
Slab:             7408 kB
CommitLimit:    779576 kB
Committed_AS:    28588 kB
PageTables:        332 kB
VmallocTotal: 4294828032 kB
VmallocUsed:      2156 kB
VmallocChunk: 4294825092 kB
```

Explanations of the fields shown in Example 5-4 on page 54 are as follows:

- MemTotal

  The amount of RAM memory assigned to Linux, not including memory used by the kernel. In Example 5-4 on page 54, Linux runs in a 128 MB virtual machine. However, only 119360 KB is available for user memory, buffers, and cache.

- MemFree

  Reports the amount of memory currently not in use by Linux.

- Buffers

  Reports the amount of memory allocated to file I/O buffers. Nowadays this is not meaningful.

- Cached

  Reports the amount of memory used in the page cache without the part reported in SwapCached.

- SwapCached

  Memory that was swapped out, was later swapped back in. It still resides in the swapfile. In case this memory is needed, pages do not need to be swapped out a again.

- Active

  Reports the amount of page cache and buffer recently used. This memory is usually not reclaimed for other use.

- Inactive

  Reports the amount of page cache and buffer cache not recently used. This memory can be reclaimed for other purposes.

- HighTotal/High Free

  Reports the total and free amount of memory that is not mapped into the kernel space

- LowTotal/Low Total

  Reports the total and free amount of memory that is directly mapped into the kernel space.

- SwapTotal/SwapFree

  Reports the total amount of swap space available and of free swap space.

- Dirty

  Reports the amount of memory that has to be written back to permanent media because of changes.

- Writeback

  Reports the amount of memory that is actively being written back to permanent media because of changes.

- AnonPages

  Reports the amount of memory that has allocated by the malloc() function and that is not backed by files.

- Mapped

  Reports the amount of memory that has allocated by the mmap() function and that is backed by files.

- Slab

  Reports the amount of memory that is used for slab cache by the kernel to hold data structures.

- CommitLimit

  If strict overcommit accounting is set (mode 2 in vm.overcommit_ratio), CommitLimit is the amount of memory that can be allocated on the system. Setting strict over-comittment is used to guarantee that one's allocated memory is available when eventually needed.

- Committed_AS

  Estimated amount of memory that has to be provided so that all processes can be allocated successfully. An out of memory situation will most likely not occur as long this much memory is provided. The amount does not reflect the memory that is already in use by all processes but that could be required at the same time.

- PageTables

  Reports the amount of memory that is dedicated to the lowest level of page tables.

- VmallocTotal

  Reports the amount of memory that is available as the vmalloc memory area.

- VmallocUsed

  Reports the amount of memory used in the vmalloc memory area.

- VmallocChunk

  Reports the amount of contiguous memory available as the vmalloc memory area.

Details about Linux Virtual Memory Management can be found in the Linux kernel documentation /usr/src/linux/Documentation/filesystems/proc.txt or at:

http://www.csn.ue.ie/~mel/projects/vm/

5.4.3 Using the vmstat command

The Linux **vmstat** command reports statistics about processes, memory, paging, I/O to block devices, and CPU usage. The command syntax is:

```
vmstat [delay [count]]
```

Where:

delay	The delay (in seconds) between updates
count	The number of updates

Example 5-5 illustrates the output from the **vmstat** command.

Example 5-5 The **vmstat** command output.

```
lnxsu2:~ # vmstat 10 7
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0  64140   3420  28572    0    0   307    61   30   73  3  2 94  1  0
 0  5  80408   1624    440   2472  553 8041  1170  8074  333  148  5  3 48 44  0
 0  1  75576  60724    352   6292 6590 5856  7458  5867  377  441  4  3  0 92  0
 0  0   8640  94780    392   7260 3226    0  3323    12  117  209  2  1 80 17  0
```

Statistics reported by the **vmstat** command are grouped by type:

- procs

  Process statistics are reported as the average number of processes in state:

  r Number of processes waiting for run time

b Number of processes in un-interruptable sleep state

- memory

  Memory statistics are reported as the average amount of memory:

  swpd Used memory size (KB)
  free Unused memory size (KB)
  buff Memory allocated to buffers (KB)
  cache Memory allocated to file system cache (KB)

- swap

  Paging statistics report average paging rates to swap devices:

  si Paging rate from swap device to memory (KBps)
  so Paging rate from memory to swap device (KBps)

- io

  I/O statistics report the average I/O rate to block devices:

  bi Number of blocks sent to a block device (blocks per second)
  bo Number of blocks received from a block device (blocks per second)

- system

  System statistics report average system activity:

  in Number of interrupts per second (including clock interrupts)
  cs Number of context switches per second

- cpu

  CPU statistics report average utilization (as a percentage) of the CPU:

  us Time spent in user mode

  sy Time spent in kernel mode

  id Time spent idle (includes I/O wait time until kernel 2.5.41)

  wa Time spent waiting for I/O (valid since kernel 2.5.41)

  st Time spent waiting for a physical CPU, also known as steal time (valid since kernel 2.6.11)

Note: Be aware that when Linux runs as a z/VM guest, the numbers reported by Linux utilities, such as **`vmstat`**, assume that the guest owns 100% of the system resources. In reality, these resources are shared by all virtual machines running in the z/VM image. Linux resource counters report values relative to the virtual machine in which they operate. Distributions including kernel levels 2.6.11 and newer are more aware of running on top of a virtualization layer and are able to report steal time with the latest tools versions (top, atop, vmstat, z/VM Performance Toolkit).

6

Tuning memory for z/VM Linux guests

In this chapter, we discuss tuning memory and storage for z/VM Linux guests based on kernel 2.6. We give you memory tuning recommendations and show you how to use new technology on the Linux kernel to reduce memory usage. Topics include:

- Memory tuning recommendation
- What is new on z/VM 5.3
- Exploiting the shared kernel
- Execute-in-place technology
- General memory tuning for guest systems

© Copyright IBM Corp. 2003, 2008. All rights reserved.

6.1 Memory tuning recommendations

In this chapter we consider how to tune memory for Linux guests. Storage recommendations are very useful in virtualized environments, especially when the Hipervisor can overcommit resources. z/VM is the most functionally capable software hipervisor operating system in existence, and fine tuning in all of the key resource areas, such as memory, is recommended for best overall system performance.

We now consider how to reduce system overhead, and then review functions introduced in Linux to optimize memory resources.

6.1.1 Reduce operational machine size

The memory footprint of servers should be minimized. There are several opportunities for reducing the size of operational memory:

- Eliminate unneeded processes.

  This might sound intuitive, but it needs to be done. Processes such as cron should be eliminated when they do not perform useful functions.

- Divide large servers into smaller specialized servers.

  Service machines can be tuned explicitly to perform a function. Linux servers running many applications do not have these controls and can be difficult to manage. Separating functions into smaller Linux guests can be done. This can make the servers easier to tailor. This is not a recommendation to separate each and every server, since a separated guest needs a certain amount of memory to run its own Linux operating system.

- Reduce machine size.

  Minimize the virtual machine size to the lowest possible amount. Choosing this amount requires research and testing. Recommendations regarding the sizing of the Linux guest can be found in 5.3, “Sizing of Linux virtual memory” on page 50.

- Use a virtual disk for a swap device.

  Using a VDISK with the DIAGNOSE access method as a swap device reduces the performance penalty of swapping. This recommendation is applicable only if you have a large amount of the real memory available on System z. Using virtual disk for swap, reduce the I/O generated by swap on real device, but at the same time increase the total amount of storage used. This tip is useful when a high paging rate comes from Linux and z/VM. In this situation is possible to experience effects of double paging (z/VM page out guest storage already swapped out by Linux) and use of VDISK as a swap device can help to reduce it.

6.1.2 Reduce infrastructure storage costs

Several opportunities for reducing infrastructure storage costs exist:

- Use the DIAGNOSE driver for the DASD and MDC record cache.

  Using a record-level minidisk cache reduces the amount of storage required for MDC. This requires the DIAGNOSE driver. For more information refer to *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8289.

- Use Cooperative Memory Management 1 (CMM1).

  If you are running Linux under z/VM you can exploit the Cooperative Memory Management 1 functions, providing a better storage allocation between guests. For more information refer to *z/VM Performance*, SC24-6109.

- Use shared kernel or execute-in-place technology.

  These new technologies can be used to reduce the total amount of storage needed to manage Linux instances. Remember that these solutions are experimental and you do not have complete support for it. For more information refer to 6.3, “Exploiting the shared kernel” on page 65, and 6.4, “Execute-in-place technology” on page 71.

6.2 Storage enhancements

Because storage is limited, every option should be taken to periodically control storage subsystem. In this chapter we show hot to control storage allocation and what is new in z/VM 5.3.

Changes to page table allocation in z/VM V5.3 allow z/VM to support significantly more real memory, up to 256 GB, twice the size supported by z/VM V5.2, with up to 20 terabytes (TB) of total virtual memory in use by guests. The actual amount of usable real and virtual memory is dependent on the amount of real memory in the z/VM logical partition, on the hardware server model, firmware level, and configuration, and on the number of guests and their workload characteristics. This can benefit your environment when you plan to have a large amount of real memory, and may help reduce or eliminate the need to spread large workloads across multiple z/VM LPARs.

Enhancements to the management of contiguous frames may also reduce memory management overhead and improve overall performance. Better z/VM management of real memory can benefit you when experience memory constrains.

In detail, z/VM 5.3 includes important enhancements to CP storage management. Page management blocks (PGMBKs) can now reside above the real storage 2 GB line, contiguous frame management has been further improved, and fast available list searching has been implemented.

These improvements collectively resulted in improved performance in storage-constrained environments, greatly increased the amount of in-use virtual storage that z/VM can support, and allowed the maximum real storage size supported by z/VM to be increased from 128 GB to 256 GB.

You can check improvement results by reading the z/VM Performance Report available at:

`http://www.vm.ibm.com/perf/reports/zvm/html/530stor.html`

6.2.1 Improved performance for storage-constrained environments

The z/VM 5.3 storage management changes have resulted in improved performance relative to z/VM 5.2 for storage-constrained environments. This is illustrated by the measurement results provided in this section.

This performance data is measured with an Apache workload. This workload consists of a Linux client application and a Linux server application that execute on the same z/VM system and are connected by a guest LAN. The Linux server is an Apache Web server that serves

URL files. The client performs a benchmark using a workload generator able to get random pages on the Apache Web server.

Figure 6-1 shows the comparison data in term of percentage improvement between z/VM 5.3 and z/VM 5.2. Each series is a different configuration of z/VM systems reproduced both in z/VM 5.2 and z/VM 5.3. Table 6-1 shows the configuration used.

Table 6-1 Test cases

Configuration	**1**	**2**	**3**	**4**
Real Storage GB	3	64	64	128
Contention	High	Low	Medium	Medium
Expanded Storage GB	4	2	2	2
Processors	3	3	3	6
Apache client/server	2/12	3/6	3/6	4/13
Apache server virtual size GB	1	10	10	10

Figure 6-1 shows small performance improvement taken on configuration 2. This is due to a very low contention used in this test. By increasing contention you can observe how z/VM 5.3 performs better in terms of throughput and CPU usage. Figure 6-1 shows the percentage improvement of z/VM 5.3 versus z/VM 5.2.

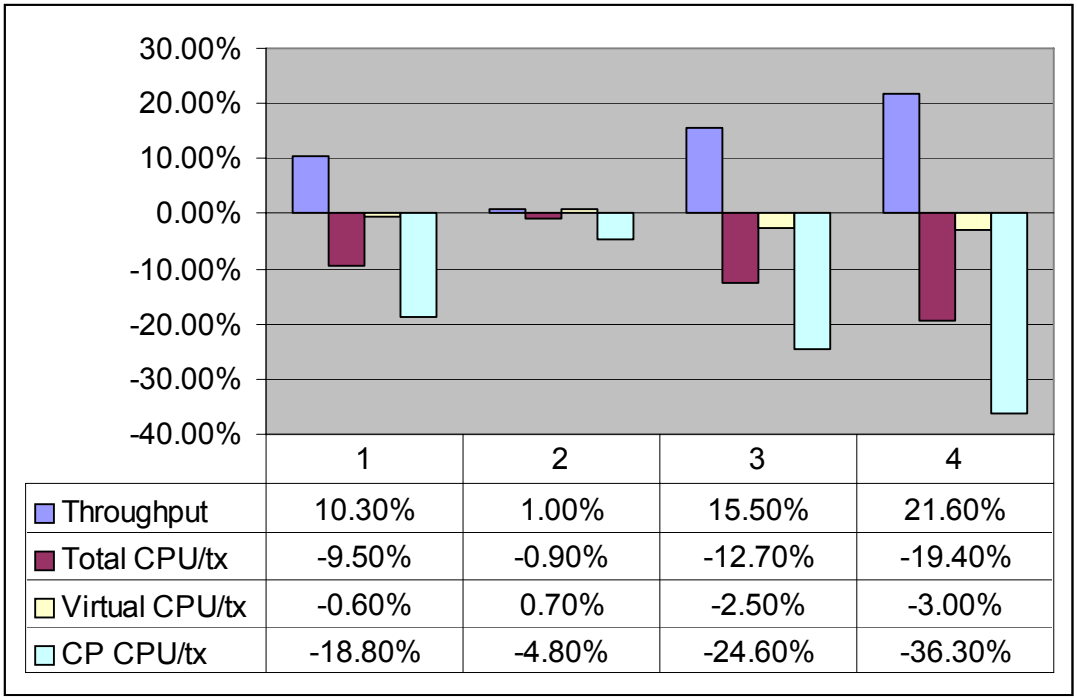


Figure 6-1 z/VM 5.3 performance improvement in storage contention situation

6.2.2 Maximum in-use virtual storage increased

Before any page can be referenced in a given 1-megabyte segment of virtual storage, CP has to create a mapping structure for it called PaGe Management BlocK (PGMBK), which is 8 KB in size. Each PGMBK is pageable but must be reside in real storage whenever one or more of the 256 pages in the 1 MB virtual storage segment it represents resides in real storage.

With z/VM 5.3 this page management block can reside above the 2 GB line so that z/VM can now support much larger amounts of in-use virtual storage.

You can see that z/VM 5.3 is not using PGMBK under the 2 GB line by looking at DSPACESH report provided by FCX123 panels in Performance Toolkit. Figure 6-2 shows our test environment.

```
FCX134      CPU 2094  SER 2991E  Interval 10:07:56 - 10:08:56    Perf. Monitor
                                          <------------------Number of Pages------------------>
Owning                                 Users       <--Resid--> <-Locked--> <-Aliases->
Userid    Data Space Name              Permt Total Resid R<2GB  Lock L<2GB Count Lockd XSTOR  DASD
>System<  --------                        0   674k  1521     1     0     0     1     0    16     5
SYSTEM    FULL$TRACK$CACHE$1              0   524k     0     0     0     0     0     0     0     0
SYSTEM    FULL$TRACK$CACHE$2              0   524k     0     0     0     0     0     0     0     0
SYSTEM    ISFCDATASPACE                   0   524k     0     0     0     0     0     0     0     0
SYSTEM    PTRM0000                        0  1049k 10630     0     0     0     0     0    60     0
SYSTEM    REAL                            0  1049k     0     0     0     0     0     0     0     0
SYSTEM    SYSTEM                          0   524k     9     9     0     0     9     0    32     1
SYSTEM    VIRTUAL$FREE$STORAGE            0   524k     5     1     0     0     0     0    19    37
```

Figure 6-2 FCX134 performance toolkit

6.2.3 Maximum supported real storage increased to 256 GB

With z/VM 5.3 you can manage up to 256 GB of real storage. This update make z/VM 5.3 more scalable than z/VM 5.2, as shown in Figure 6-3 and Figure 6-4 on page 65. Figure 6-3 shows performance data in terms of transaction per second and the amount of total CPU spent per transaction. You can see z/VM 5.3 performance scale up increasing real storage and the number of processors.

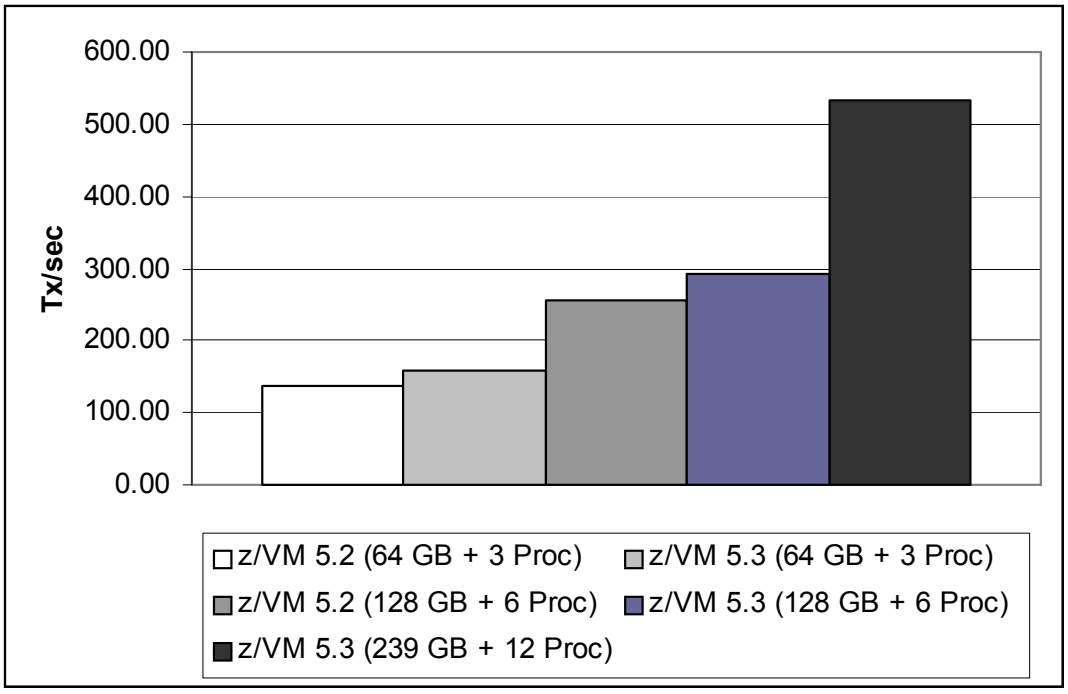


Figure 6-3 Storage and processor scaling

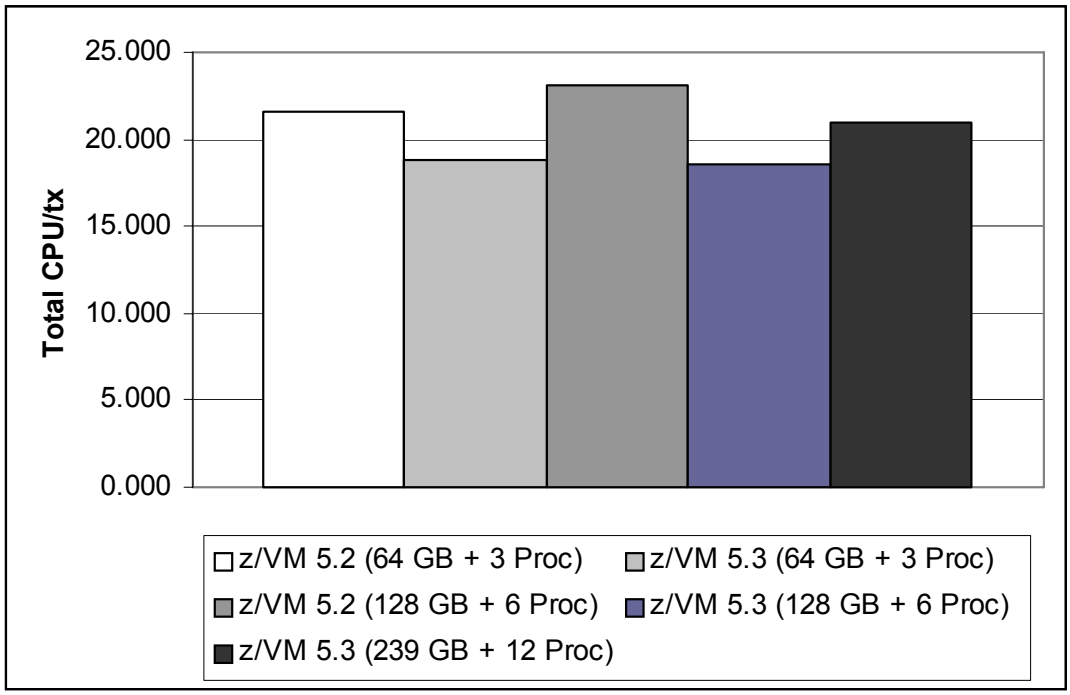


Figure 6-4 CPU usage per transaction

Note: For more information about performance data, refer to:

`http://www.vm.ibm.com/perf/reports/zvm/html/530stor.html`

6.3 Exploiting the shared kernel

We believe that it is important for the scalability of Linux on z/VM to share the Linux kernel in NSS. The ability to share the Linux kernel will be released and supported later in other service packs for SUSE distribution.

Note: To create a Linux system that utilizes NSS support for other SUSE releases or other distributions, you need to recompile the Linux kernel. Be aware that your distributor might not support systems running with a custom compiled kernel. Because this might be the case, NSS support should be considered experimental. Depending on the level of support required, this approach might not be appropriate for your installation.

To create a Linux NSS in SLES 10 with SP1 you need to:

1. Define a skeletal system data file (SDF) for the compiled kernel using the CP DEFSYS command.
2. Save the NSS-enabled kernel into the SDF using the CP SAVESYS command.

For all other distributors or release you need to:

1. Compile the Linux kernel with the appropriate configuration options for NSS enabled.
2. Define a skeletal system data file (SDF) for the compiled kernel using the CP DEFSYS command.

3. Save the NSS-enabled kernel into the SDF using the CP SAVESYS command.

The kernel configuration option CONFIG_SHARED_KERNEL is used to enable the kernel code and data in memory segments. This is necessary because the shared kernel code must be protected against updates from the Linux images that use that kernel. In S/390 architecture, protection is assigned on a segment basis (each segment is 1 MB in size).

Note: When Linux is running in dedicated memory (for example, on an Intel® server or in a S/390 LPAR), this option would not be used. It would only waste some of the memory dedicated for use by that server. However, when running Linux in a virtual machine, there is very little cost involved with virtual memory that is never used. It does not have to be provided by z/VM either (the lost memory below 3 MB can be compensated for by giving the virtual machine slightly more memory).

Figure 6-5 shows how a kernel map can look with new releases available, for example, with SUSE 10.

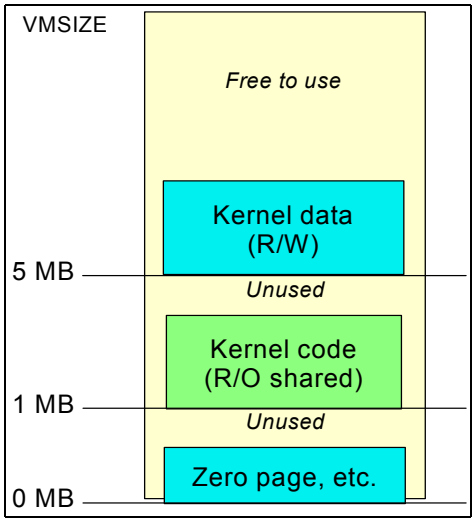


Figure 6-5 Linux memory map with shared kernel

6.3.1 Building an NSS-enabled Linux kernel

This procedure is intended for distribution that currently does not support the NSS in the kernel. If you are running a Linux with SLES10 SP1, can refer to 6.3.2, “Defining a skeletal system data file for the Linux NSS” on page 68. To build a Linux kernel for System z:

1. Obtain the kernel source code.
2. Obtain and apply the zSeries-specific and S/390-specific patches and object code only (OCO) modules.
3. Configure and build the kernel.
4. Copy the OCO modules to the appropriate locations in the file system.
5. Copy the new kernel to the /boot directory and run the **`zipl`** command.

Details about building a Linux kernel for zSeries can be found in *Linux on IBM eServer zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP-3596, at:

`http://www.ibm.com/redbooks/abstracts/redp3596.html`

For NSS support, we need to enable the CONFIG_SHARED_KERNEL option. Using the **make menuconfig** command, this option can be enabled by selecting **General Setup → VM shared kernel support**. Example 6-1 shows the main panel with the General Setup option.

Example 6-1 Main panel: make menuconfig

```
Code maturity level options  --->
Loadable module support  --->
Processor type and features  --->
General setup  --->
SCSI support  --->
Block device drivers  --->
Multi-device support (RAID and LVM)  --->
Character device drivers  --->
Network device drivers  --->
Miscellaneous  --->
Networking options  --->
File systems  --->
Kernel hacking  --->
---
Load an Alternate Configuration File
Save Configuration to an Alternate File
```

In the General setup menu shown in Example 6-2, select the **VM shared kernel support** option.

Example 6-2 General setup: VM shared kernel support

```
[*] Fast IRQ handling
[*] Process warning machine checks
[*] Use chscs for Common I/O
<M> QDIO support
[ ]    Performance statistics in /proc
[*] Builtin IPL record support
(vm_reader) IPL method generated into head.S
[*] Networking support
[*] System V IPC
[ ] BSD Process Accounting
[*] Sysctl support
<*> Kernel support for ELF binaries
< > Kernel support for MISC binaries
[ ] Show crashed user process info
[*] Pseudo page fault support
[*] VM shared kernel support
[*] No HZ timer ticks in idle
[ ]   Idle HZ timer on by default
```

One should not underestimate the cost of compiling the kernel. On a zSeries CPU, a complete build of the kernel may well use some 10 to 15 minutes of CPU time. If you worry about the cost of an idle Linux virtual machine, a trimmed down Linux image can run idle for more than a month on the amount of CPU cycles the kernel build takes.

Important: At least with the Linux 2.4.7 kernel sources, you should run a **make clean** after you changed the CONFIG_SHARED_VM option. There appears to be a problem with the build process in that it does not pick up the change.

6.3.2 Defining a skeletal system data file for the Linux NSS

We examine the generated System.map file (shown in Figure 6-5 on page 66) to identify the zero page, shared kernel code, and non-shared kernel data regions in the constructed kernel. See Example 6-3.

Example 6-3 Portions of the System.map to determine NSS configuration

```
00000000 A _text
00000298 t iplstart
00000800 T start
00010000 t startup
00010400 T _pstart
00011000 T s390_readinfo_sccb
00012080 T _pend

00100000 T _stext
00100100 t rest_init
00100148 t init

004ef190 r __param_maxchannels
004ef1b8 r __param_format
004ef1e0 R __stop___param
00500000 A _eshared
00500000 D init_mm
00500310 D init_task

005ca000 d per_cpu__flow_flush_tasklets
005ca100 d per_cpu__rt_cache_stat
005ca140 d per_cpu____icmp_socket
005ca148 A __per_cpu_end
005cb000 A __bss_start
005cb000 A __init_end
005cb000 B system_state
005cb008 B saved_command_line
005cb010 B __per_cpu_offset
```

We find the page range for each region based on the symbols that indicate the start and end of the region.

Note: The address we use to locate the end of the region is one byte past the end of the region. To compensate, we subtract one byte in the calculation. Because pages are 4 K in size, we calculate the page range by dropping the last three nibbles from the address.

From the System.map file, we identify:

- The region extending from _text to _pend is the zero page area. Using addresses 0x00000000 and 0x00012080, we find that the page range is 0–12.
- The region extending from _stext to the symbol *immediately before* _eshared (__stop___param) is the shared kernel code. Using addresses 0x00100000 and 0x004ef1e0, we find that the page range is 100-4ef.
- The region extending from _eshared to _bss_start is the non-shared kernel data. Using addresses 0x00500000 and 0x005cb000, we find that the page range is 500-5cb.

From these values, we construct the following DEFSYS command:

```
DEFSYS SLES10 0-12 EW 100-4ef SR 500-5ca EW MINSIZE=24M MACHMODE XA,ESA
```

Parameters to the command are:

- SLES10

  The name to identify the NSS on IPL. By using different NSS names, you can have different versions of the kernel saved as NSS on a VM system. The IPL command identifies the NSS to be used by the virtual machine. Use this ability with care. Sharing is most efficient (both in memory and system administration) when many virtual machines share the same NSS.

- 0-12 EW

  The page range for the zero page region. This region is copied to each virtual machine in exclusive write mode (EW).

- 100-4EF SR

  The page range for the shared kernel code. Each virtual machine uses a shared, read-only copy (SR).

- 500-5CA EW

  The page range for the private kernel data. Each virtual machine uses an exclusive write copy.

- MINSIZE=24M

  The minimum virtual machine size to use the NSS (only to prevent an IPL in a virtual machine where it will not fit anyway).

- MACHMODE XA,ESA

  The NSS can be IPLed in an XA or Enterprise Systems Architecture (ESA) virtual machine.

6.3.3 Saving the kernel in the Linux NSS

The NSS-enabled kernel is saved into the skeletal SDF using the SAVESYS command. As an example, we use the SAVELX EXEC script from the How To Use VM Shared Kernel support page at:

http://www.vm.ibm.com/linux/linuxnss.html

We modify the script to use the DEFSYS command parameters specific to our kernel, as shown in Example 6-4.

Note: The original values used for the DEFSYS command work when creating the SDF. However, these values create a larger NSS than required (and therefore require more time to IPL).

Example 6-4 SAVELX EXEC to save a Linux NSS

```
/* SAVELX EXEC */
/* get the system name and device to ipl */
parse arg lxname devnum
lxname = strip(lxname)
devnum = strip(devnum)
/* figure out the line end character */
'pipe cp q term | var termout'
parse var termout one myend three
myend = strip(myend)
/* figure out the storage size */ 'pipe cp q v stor | var storout'
parse var storout one two storsize
/* construct the defsys command */
DODEF = 'DEFSYS' lxname '0-12 EW 100-4EF SR 500-5CA EW MACHMODE XA,ESA'
dodef = dodef 'MINSIZE=' || storsize
say dodef
/* define the saved system */
dodef
/* arrange to stop the ipl processing at the appropriate spot, */
/* at which point a savesys will be issued */
SETSAVE = 'TRACE I R 010000 CMD SAVESYS' lxname
setsave = setsave || myend 'TRACE END ALL'
say setsave
setsave
doipl = 'i' devnum
say doipl
/* all set, issue the ipl */
doipl
exit
```

SAVELX EXEC requires two input parameters:

1. The name of the Linux NSS. In our example, we use the name SLES10.
2. The DASD device on which the NSS-enabled kernel resides. This DASD device should be defined for each virtual machine using the Linux NSS.

Assuming that the NSS-enabled kernel resides on virtual device 201, we save Linux using:

```
SAVELX SLES10 201
```

6.3.4 Changing Linux images to use the shared kernel in NSS

To boot the Linux NSS in a virtual machine, issue the CP IPL command using the name of the Linux NSS. In our example, the NSS is named SLES10:

```
IPL SLES10
```

Be aware that although the NSS-enabled kernel is not booted from DASD, DASD device numbers for virtual machines using the NSS must match the device numbers used when the NSS was created. For example, if the NSS-enabled kernel was created using a 201 disk as

the root file system device, virtual machines using the NSS must define a 201 disk with a root Linux file system.

6.4 Execute-in-place technology

You can keep down memory requirements and improve your performance of virtual Linux using discontiguous saved segment (DCSS). In a virtualized environment, Linux images may need the same data at the same moment. Doing that, the same data might get loaded into memory several times. A major part of memory required by Linux is used for binary application files and for shared library files. This technology helps you provide a way to share memory for some application using a discontiguos saved segment managed by z/VM.

When you plan to share data consider that:

- Application files can only be shared by Linux instances that use the same version of an application.
- Shared data must be read-only.
- Applications and libraries that are frequently used by numerous Linux instances are good candidates.

DCSS can be defined above Linux guest storage or in a storage gap. Usually, using DCSS in a storage gap is the preferred placement for a 64-bit Linux guest. Linux can access a DCSS through the execute-in-place file system (xip2). The functionality of this technology is easy to understand.

Linux load shared library and application files through the mmap() operation. mmap() maps the contents of a file into the application's address space. The xipl file system performs this operation by mapping the contents of the DCSS into the application's address space while other file systems use a page cache. This feature is called execute-in-place because the file contents are not physically copied to a different memory location. With execute-in-place technology, Linux can:

- Access files and libraries without I/O operations (which increases overall performance).
- Run an application directly in a shared memory segment (which saves memory).

6.4.1 Setting up a DCSS

In this chapter we provide an example of how to use a DCSS. If you are planning to use this technology, refer to documentation available at:

`http://www.ibm.com/developerworks/linux/linux390/october2005_documentation.html`

Plan for a DCSS

The directory we are going to use for the experiment is the /usr. The **`du -sk`** command shows how big the directory is. This is important when we define the DCSS. In our example the directory is 1035264 KB. We need to add some extra space in order to manage meta data and future space for software updates. We increase our directory up to 1258291 KB (1.2 GB).

We are now ready to set up our DCSS. Refer to Table 6-2 to check the maximum DCSS size allowed.

Table 6-2 Maximum DCSS size

Kernel	DCSS location	Maximum DCSS size
64-bit	Storage gap	1919 MB
64-bit	Above guest storage	2047 MB
31-bit	Either location	1960 MB

In our example we chose to assign DCSS above guest size. We now determine the start and end addresses of our DCSS. This addresses must be on a page boundary (4 KB) and in hexadecimal notation. In our example:

- Start address: 512 MB = 0x20000000
- End address: 512 MB + 1011 MB - 1 MB = 0x5F2FFFFF
- Start frame: 0x20000
- End frame: 0x5F2FF

Before the Linux kernel can use the DCSS above the guest storage, it must be aware of the extended address space that covers the DCSS. To do that, simply add a mem=<value> on your kernel parameter file, run zipl, and reboot it. In our example we add:

```
mem=1523M
```

Create over-mounting script

We are chose to use an entire directory. This means that we need to over-mount our DCSS on the directory that we are including in the segment. This directory resides on the root partition, so we need a script to be run before system startup.

IBM provides a sample script to be tailored, called xipinit.sh. Example 6-5 shows the xipinit.sh script tailored for our purpose. The changes we made are in bold.

Example 6-5 xipinit.sh

```
#!/bin/sh
#
# xipinit.sh, Version 2
#
# (C) Copyright IBM Corp. 2002,2005
#
# /sbin/xipinit: use read only files from another file system
#
# default options
# internals
#set -v
#FAKE=echo
#mount point of xipimage
ROMOUNT="/mnt"
#name of xipimage
XIPIMAGE="USRXIP"
#name of device node
RODEV="/dev/dsccblk0"
RODIRS="/usr,"
# make sure it ends with ,
RODIRS="$RODIRS",
```

```
mount -t sysfs none /sys
if [ ! -e /sys/devices/dcssblk ]; then
echo "xipinit: loading dcssblk module"
/sbin/modprobe dcssblk
fi
echo $XIPIMAGE > /sys/devices/dcssblk/add
# mount ro file system to its mount point
echo "xipinit: mounting read-only segment"
$FAKE mount -t ext2 -o ro,xip "$RODEV" "$ROMOUNT"
# bind mount all ro dirs into rw file system
while [ -n "$RODIRS" ] ; do
dir="${RODIRS%%,*}"
RODIRS="${RODIRS#*,}"
test -d "$dir" || continue
echo "xipinit: binding directory" $dir
$FAKE mount --bind "$ROMOUNT/$dir" "$dir"
done
umount /sys
# run real init
$FAKE exec /sbin/init "$@"
```

We have modified three parameters:

- ROMOUNT

  This is the directory where DCSS should be mounted. (For the first run we use /mnt.)

- XIPIMAGE

  This is the name of DCSS that we use. (@e use USRXIP.)

- RODIRS

  This is the directory that we include on the DCSS.

Creating a DCSS

In order to create the segment, we use the block device driver. To use it, the Linux instance must have class privilege E. These are steps necessary to build the DCSS:

1. Shut down your Linux image and IPL the CMS.
2. Define the DCSS with the command:

   ```
   DEFSEG USRXIP 20000-5F2FF SR
   ```

3. Define the storage guest to cover the entire DCSS (increasing to 2 GB covers any possible DCSS):

   ```
   DEFINE STOR 2048M
   ```

4. Save the DCSS (this command can take a long time to run):

   ```
   SAVESEG USRXIP
   ```

Copying data to the DCSS using the DCSS block device driver

Following this procedure is necessary because data from /usr must be copied inside the new file system. You cannot use the xip2 file system for this step because it is a read-only file system. We use ext2, which is similar and compatible in terms of structure. After used in a read/write mode, you can unmount and remount it with xip options.

To copy data into DCSS:

1. Load the DCSS module:

   ```
   # modprobe dcssblk
   ```

2. Create the node for the /dev directory structure (the major number is indicated in /proc/devices):

   ```
   # mknod /dev/dcssblk0 b 252 0
   ```

3. Load the DCSS created:

   ```
   # echo "USRXIP" > /sys/devices/dcssblk/add
   ```

4. Change the access mode from share to exclusive-writable:

   ```
   # echo 0 > /sys/devices/dcssblk/USRXIP/shared
   ```

5. Create an ext2 file system on it:

   ```
   # mke2fs -b 4096 /dev/dcssblk0
   ```

6. Mount the file system on /mnt:

   ```
   # mount /dev/dcssblk0 /mnt/
   ```

7. Copy the contents of /usr to /mnt:

   ```
   # cp -a /usr/* /mnt
   ```

8. Save the DCSS using:

   ```
   # echo 1 > /sys/devices/dcssblk/USRXIP/save
   ```

   Do not worry if you see a message on the z/VM console like `Segment USRXIP is currently busy, it will be saved when it becomes idle`. The segment is saved when you umount the file system.

9. Umount the /mnt file system:

   ```
   # umount /mnt
   ```

Testing the DCSS

To test the DCSS created, mount the xip2 file system:

```
# mount -t ext2 -o ro,xip /dev/dcssblk0 /mnt
```

To see whether file system is mounted correctly, check the /proc/mounts file systems, as shown in Example 6-6.

Example 6-6 mount point

```
lnxsu4:/ # cat /proc/mounts
rootfs / rootfs rw 0 0
udev /dev tmpfs rw 0 0
/dev/dasdb1 / ext3 rw,data=ordered 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
debugfs /sys/kernel/debug debugfs rw 0 0
devpts /dev/pts devpts rw 0 0
/dev/dcssblk0 /mnt ext2 ro,nogrpid,xip 0 0
```

Activate execute-in-place

Before you take any steps, check whether the scripts provided in Example 6-5 on page 72 work. The script is located in the directory that we saved (/sbin).

At the end of the script, issue a **cat /proc/mount** to confirm that the directory is mounted.

Example 6-7 xipinit.sh script test

```
# /sbin/xipinit.sh
# cat /proc/mounts
udev /dev tmpfs rw 0 0
/dev/dasdb1 / ext3 rw,data=ordered 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
debugfs /sys/kernel/debug debugfs rw 0 0
devpts /dev/pts devpts rw 0 0
```

Umount the file system, and add the following line to /etc/zipl.conf in the parameter line:

```
init=/sbin/xipinit.sh
```

Save zipl and reboot your Linux system.

If the file system is mounted you can edit the script to change the /mnt directory to /usr in order to have the entire /usr file system mounted as DCSS.

In a large Linux server farm on z/VM, this technology helps to increase overall performance.

6.5 Cooperative memory management (CMM)

The CMM is one of the solutions that allows an external entity, such as the z/VM resource monitor (VMRM) or a third-party external monitor, to reduce the storage size of a Linux guest. The Linux Kernel module cmm has to be loaded to make the collaboration with the external monitor possible. The external monitors monitor the Linux guests and how these guests cope with their memory requests. Through the interfaces created with the cmm module, the Linux guest receives information about how much free memory should be available. When not enough free memory is available, the Linux guest starts to clean the page cache, the slab cache, or other memory until it meets the threshold's requirement. Under pressure, the Linux guest may start swapping.

In the documentation, in some cases CMM is referred to as CMM1.

More details can be found in *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8289-03:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/docu/l26cdd03.pdf

Or on the z/VM Web pages found at:

http://www.vm.ibm.com/sysman/vmrm/vmrmcmm.html

Prerequisites to use CMM

The module „cmm“ has to be loaded within the involved Linux guests. VMRMSVM has to be well configured and started in z/VM.

Setup of CMM

To set up z/VM to use CMM we created a VM Resource Manager (VMRM) configuration file [VMRM CONFIG A1] on the VMRMSVM user ID that contained the following statement:

```
NOTIFY MEMORY LNX001 LNX002 LNX003 LNX004 LNX005
```

Or:

```
NOTIFY MEMORY LNX00*
```

We also added this statement to get VMRMSVM messages:

```
ADMIN MSGUSER MAINT
```

This statement causes VMRMSVM messages to be sent to the MAINT user ID.

The asterisk is a wild card. After creating the configuration file on the VMRMSVM A-disk we logged off VMRMSVM and issued a xautolog VMRMSVM from user MAINT to start the VMRM server.

To stop the VMRMSVM server, log on to VMRMSVM and issue the command HMONITOR. (or FORCE VMRMSVM as user MAINT)

On the Linux servers we load the CMM kernel extension with the **modprobe** command as follows:

```
modprobe cmm
```

6.6 Collaborative memory management assist (CMMA)

Linux guests and the z/VM V5.3 control program (CP) exchange information regarding memory usage. This exchange of information allows both the guests and the z/VM host to optimize their management of memory, in the following ways:

- CP knows when a Linux application releases storage and can select those pages for removal at a higher priority or reclaim the page frames without the overhead of paging-out their data content to expanded storage or disk.
- CP recognizes clean disk cache pages, the contents of which Linux is able to reconstruct, allowing CP to bypass paging-out the data contents when reclaiming the backing frames for these pages. If Linux or its application subsequently tries to refer to the discarded page, Linux is notified that the page has been discarded and can reread the contents from disk or otherwise reconstruct them.
- In conjunction with CMMA, Host Page-Management Assist (HPMA) allows the machine to supply fresh backing page frames for guest memory when the guest reuses a previously discarded page, eliminating the need for the z/VM Hypervisor™ to intercept and resolve these host page faults.

For more details go to:

```
http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&appnam
e=GPA&htmlfid=897/ENUS207-019
```

If CMMA is enabled, the pages are flagged differently compared to a Linux system not using this feature. When the page cache gets filled, each page gets flagged as stable or as volatile. This starts already with the boot process. Volatile flagged pages can get stolen by z/VM and provided to other guests. This is not the case for the pages that are flagged stable. More

details about the Linux part can be found in *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8289.

At the time we are writing this book IBM is working with its Linux distribution partners to provide CMMA exploitation in future Linux for System z distributions or service updates.

Prerequisites to use CMMA

The prerequisites are:

- z/VM 5.2 with APAR VM63856 applied or z/VM V5.3
- z9 EC or z9 BC processors or later

Setup for CMMA

The Linux has to be IPLed with the option cmma=on. Thereupon, the kernel uses special page flags. By default, CMMA is off.

7

Linux swapping

In this chapter we examine options for setting up a Linux swap file. Topics include:

- Linux swapping basics
- Linux swap device options and comparison
- Swapping in peak situations versus swapping regularly
- Recommendations for swapping

© Copyright IBM Corp. 2003, 2008. All rights reserved.

7.1 Linux swapping basics

Over time, a Linux system uses all the available memory. When it does not need the memory to run processes, it uses all excess memory to cache data. The most obvious way to prevent this is not to give the virtual machine more memory than it needs. Unfortunately, this is not always possible. Memory requirements vary over time as processes start and stop. Consequently, the virtual machine is either too big or too small.

When the virtual machine is too small for the workload, Linux starts to swap. Pages from other processes are moved out and are stored on the Linux swap disks to make room for the pages of the process that need to run. If Linux swaps continuously, performance is affected. Occasional swapping inside Linux is not necessarily bad. The acceptable amount of swapping depends on the efficiency of the swapping mechanism.

To compare the efficiency of swap device types for Linux, we ran a number of hogmem processes in a small virtual machine. The hogmem program is very simple. The listing can be found in 7.9, “Program text for hogmem test” on page 99. It allocates the specified amount of virtual memory for the process and then sequentially accesses each page. When the amount of memory allocated by hogmem exceeds the free memory in Linux, some other pages are swapped out by Linux. When we run enough hogmem programs in parallel and allocate more virtual memory than what Linux can free up for use, constant swapping occurs (which slows down normal operation).

When we IPL the kernel in a 128 MB virtual machine, the output of command **free -m**, as shown in Example 7-1, suggests that nearly a half of that storage should be available for processes.

Example 7-1 Available memory in an idle 128 MB virtual machine

```
# free -m
             total       used       free     shared    buffers     cached
Mem:           116         55         60          0          3         24
-/+ buffers/cache:         27         89
Swap:         7043          0       7043
```

The output shows that 60 MB is not in use. Much of the buffers and cache could be reclaimed by Linux when necessary, so 87 MB is close to what we can obtain without causing much swapping.

After starting a 87 MB process, we re-examine memory usage in Example 7-2.

Example 7-2 Memory usage in 128 MB machine with 87 MB process

```
# free -m
             total       used       free     shared    buffers     cached
Mem:           116        115          1          0          0          4
-/+ buffers/cache:        110          5
Swap:         7043          6       7036
```

This shows that Linux did not give up all buffers and cache, but decided to move some things to swap instead:

- Linux obtained 59 MB from the free pool, leaving 1 MB available to satisfy sudden and urgent memory requests.
- The remaining 28 MB was obtained from buffers/cache and by memory freed by swapping.

As shown in Example 7-3, **vmstat** reports very little swapping while the process runs.

Example 7-3 Monitoring swap activity

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-------cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0  56392   4344  28680    0    0    49    13   13   24  1  2 97  0 0
 0  0      0  56364   4344  28680    0    0     0     8   16   10  0  0 100  0 0
 0  0      0  56364   4344  28680    0    0     0     0    6    3  0  0 100  0 0
 0  0      0  56364   4344  28680    0    0     0     0    6    5  0  0 100  0 0
 0  0      0  56364   4344  28680    0    0     0     0    6    7  0  0 100  0 0
 1  0      0  12724   4352  28672    0    0    16    44   35   28 19  3 78   0 0
 1  1  40884   1476    116   1560    0 62316  2580 62364 1868 383 16 18  0  66 0
 0  1  62316   4324    128   2764  192    0  1768     0  103   37  1  2  0  97 0
 0  3  62316   1560    128   2960 3736    0  3736     0  156  301  2  1  0  97 0
 0  3  62316   1680    128   3056 1372    0  1372     0  176  310 25  2  0  73 0
 0  1  81708  11240    104    664 3092 34532  3652 34580 2339 850  1  5  0  93 0
 1  1  81708   5808    104   1120 5176    0  5608     0  703  730 32  5  0  62 1
 1  0  81708   5500    104   1304  332    0   332     0  169  135 97  0  0   3 0
 1  0  81708   5140    108   1720   96    0   464     0  113   24 100 0  0   0 0
 1  0  81708   5140    108   1720    0    0     0     0  105    9 99  1  0   0 0
 1  0  81708   5140    108   1720    0    0     0     0  105   13 100 0  0   0 0
 1  0  81708   5140    108   1720    0    0     0     0  107    9 98  1  0   0 1
```

Linux swap cache

Even the implementation of the swap cache in Linux tries to reduce the I/O burden. Linux starts swapping from time to time. The copy of the swapped-out page is held on the swap device or file after the page is eventually switched, so if the page has to be swapped out for the second time there are two options. If the page was not changed (is not dirty), it could simply be discarded. If the page was already changed (is dirty), it has to be brought out a second time. The page already has a location in the swap space assigned that reduces the overhead for assigning a new location.

Note: The Linux swap cache sometimes causes misleading swap-in and swap-out page amount information in monitoring commands. This is the case when using tools like **vmstat**. The following paragraphs often show more pages swapped in than swapped out.

7.2 Linux swap options

Linux on System z has many options when its comes to choosing the appropriate swapping device. Often installations use VDISK when running on z/VM, and ECKD™ DASD when running in an LPAR without z/VM. There are new options like the DIAGNOSE support for 64-bit DASD (supported on z/VM 5.2 or later), which was previously only available in 31-bit. When choosing a swapping device, the most important thing to consider is whether swapping is expected to be a normal part everyday processing, or whether swapping occurs only occasionally. But, if an application starts swapping, the performance is dependent on the ability to transfer pages to and from the swap device quickly.

With three different types of disk, and four different drivers, we have seven valid combinations. We measured the swap speed of five options, which are shown in Table 7-1.

Table 7-1

Driver/device type	DASD	VDISK	SCSI disk
ECKD	Default	N/A	N/A
DIAGNOSE[a]	Alternative	Alternative	N/A
FBA	Alternative	Default - ldl format	Alternative
FCP	N/A	N/A	Default

a. DIAGNOSE is only available if Linux is running as a z/VM guest.

7.3 Swapping to DASD

We first set the baseline for our benchmark by measuring the performance of the extended count key data (ECKD). To cause more swapping, we must push a bit harder. If pushed hard enough, we can obtain a fairly constant swap rate that can be studied with the different measurement tools. In this test we used one or two 3390-9 DASDs on an ESS-800.

7.3.1 Swapping with ECKD discipline to DASD

The output in Example 7-4 shows what happens when we increase the virtual memory to 128 MB. Because our program hogmem walks through the allocated memory sequentially, we force all pages from swap into memory and out to swap again.

Example 7-4 Swapping during memory usage test

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0   2208   3084  21168    0    0  1340    37  117  206  5  7 61 28  0
 0  0      0   2208   3084  21168    0    0     0     0   10    4  0  0 100  0  0
 0  0      0   2208   3084  21168    0    0     0     0    9    7  0  0 100  0  0
 0  0      0   2208   3084  21168    0    0     0     0    4    3  0  0 100  0  0
 0  5   4544   1292    272   2784    0 13544  3012 13544  145  324  8  7 30 54  1
 0  4  49224   1108    104   1848  416 35700  9028 35800  801  753 15 16  0 69  0
 1  1  80940   1808    104    940  412 31780  4428 31780  209  492 14 14  0 71  1
 0  3 111156   1060    108   2528 4532 30288 14720 30288 1515  781 13 15  0 72  0
 0  1 136776  11432    104    348 3332 29648  5544 29648  295  410  1  5  0 93  1
 0  2 137436   1052    108   1736 20820  796 22876   796  809 1424  8  6  0 86  0
 0  1 138368   2968    108    192 10008 30364 13600 30364 2116  812  4 11  0 86  0
 0  2 138368   1224    104   1808 18776   44 21700    44  864 1317  7  6  0 88  0
 0  2 138368   1648    104    444 13564 31916 15052 31916 1328 1013  4 11  0 84  1
 0  3 138368   1064    104   2716  1384    0  3004     0  134  148  1  2  0 97  0
 0  2 138368   1224    104    256 24204    0 24720     0 1976 1604  8  6  0 86  0
 0  1 138368   6268    108    668 8112 31748  9748 31748  456  662  3 10  0 88  0
```

In this example, we note that the swap rate is reported in columns si (memory swapped in from disk) and so (memory swapped out to disk). These values are reported in terms of the number of 1 KB blocks transferred. The **vmstat 1** catches values every second and illustrates the non-steady swapping. A **vmstat 60** command would have displayed the number of

swapped-in and swapped-out pages as being more constant over time, generating an average value per minute.

Example 7-5 shows results for swapped-in and swapped-out pages retrieved by the performance toolkit. Unlike the results in Example 7-4 on page 82, the numbers are aggregated over a period of one minute. The swap rate is the sum of swapped-in and swapped-out pages. We measured a swap rate of approximately 4 MBps for the ECKD DASD combination. If we compare the swapped-in and swapped-out values we see that the first minute's swap-out is higher than is swap-in. The pages are written to disk and not all of them have to be brought back. Starting with the fifth minute we see fewer pages written to the swap devices. This is the effect of the swap cache. Some of the pages are already on the swap device. There is no need to transfer it again to the swap device as long they are not dirty pages.

Example 7-5 Memory at swapping using ECKD DASD

```
<-Memory(MB)->  <------- Swapping -----> <-BlockIO->
Interval   <-- Main --->  <-Space (MB)> <Pgs/sec>  <--kB/sec->
  Time   M_Total %MUsed   S_Total %SUsed  In  Out   Read  Write
16:41:05  53.8     95.9    7043       0    0    0  0.138  6.138
16:42:10  53.8     98.1    7043     1.9 1818 2205   8825   8823
16:43:11  53.8     97.7    7043     1.9 2488 2303  12018   9224
16:44:10  53.8     98.2    7043     1.9 2715 2586  12818  10346
16:45:10  53.8     97.9    7043     1.9 2401 2153  11196   8620
16:46:11  53.8     98.1    7043     1.9 2087 1992   9671   7968
16:47:11  53.8     95.6    7043     1.9 2059 1951   9727   7807
16:48:10  53.8     97.3    7043     1.9 2012 1870   9404   7483
16:49:10  53.8     97.0    7043     1.9 2021 1844   9371   7379
16:50:10  53.8     97.9    7043     1.9 2057 1959   9636   7839
16:51:11  53.8     97.9    7043     1.9 2184 2077  10244   8310
```

In Example 7-6 we note that more than 90% of the time the CPU is waiting for I/O completion. So we conclude that the throughput of our test application is down to 5% or less due to swapping.

Example 7-6 CPU utilization at swapping using ECKD DASD

```
Interval   <-------CPU Utilization (%)----------->
Time       TotCPU User  Kernel IOWait Idle  Stolen
16:41:05    0.4   0.3   0.1      0.1  99.5      0
16:42:10    8.7   3.5   4.2     69.3  21.9    0.2
16:43:11   10.4   4.2   4.9     89.4     0    0.2
16:44:10   10.8   4.2   5.2     89.0     0    0.2
16:45:10    9.3   3.7   4.5     90.5     0    0.2
16:46:11    8.1   3.1   4.0     91.7     0    0.1
16:47:11    8.1   3.1   4.1     91.8     0    0.2
16:48:10    7.6   3.0   3.7     92.3     0    0.2
16:49:10    7.6   3.0   3.6     92.3     0    0.1
16:50:10    7.9   3.0   3.9     91.9     0    0.2
16:51:11    8.5   3.2   4.3     91.3     0    0.2
```

We use an extreme and artificial benchmark, which is only touching lots of pages. A real application should not suffer as much from swapping, since it has to spend more time to work with the available pages. Not much time is reported waiting for involuntary wait.

Effect of the number of processes swapping to ECKD DASD

Many processes are running on Linux production systems. So, in short-of-memory situations, more than one process uses the swap service. We measure the ECKD DASD swap option running three hogmem programs in parallel, each allocating 50 MB. The swap disk is one 3390-9 DASD on an ESS-800 (Example 7-7).

Example 7-7 Memory when three processes swap using one ECKD DASD

```
<-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval   <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
18:23:10  53.8    97.3    7043    2.2  2452 2849 11386  11398
18:24:10  53.8    97.7    7043    2.2  3298 2996 15282  11986
18:25:10  53.8    94.6    7043    2.2  3205 3098 14627  12393
18:26:10  53.8    98.1    7043    2.2  2863 2727 12146  10909
18:27:11  53.8    97.8    7043    2.2  3024 2883 12830  11532
18:28:10  53.8    97.8    7043    2.2  2838 2663 12065  10652
18:29:11  53.8    98.1    7043    2.2  3094 2848 13330  11392
18:30:10  53.8    98.1    7043    2.2  3181 2848 13965  11398
18:31:10  53.8    96.9    7043    2.2  3085 2937 13096  11749
18:32:10  53.8    98.2    7043    2.2  2968 2742 12453  10969
18:33:10  53.8    98.0    7043    2.2  3028 2779 12755  11114
```

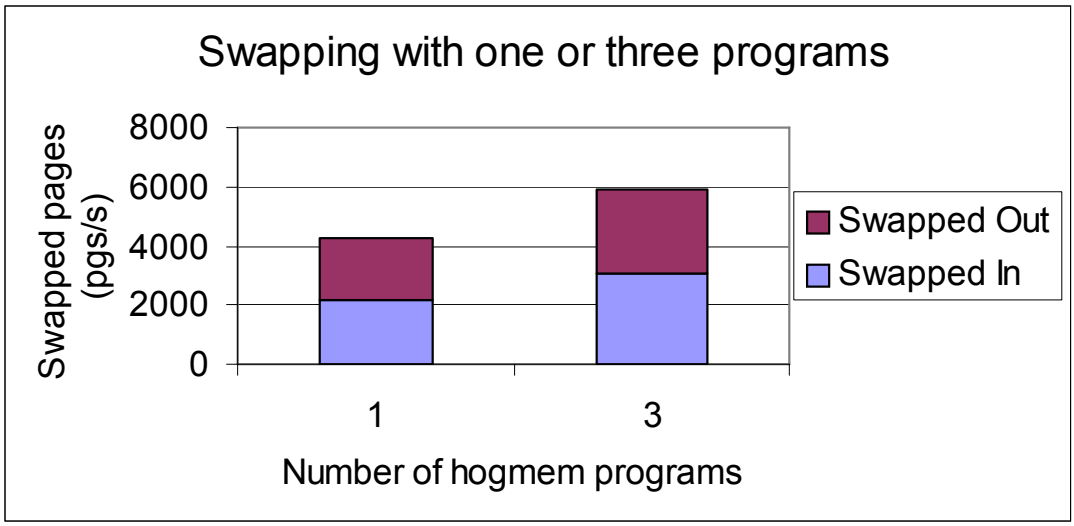


Figure 7-1 Swap rate when one or three hogmen programs swap using one ECKD DASD

Compared to the single process running (Example 7-5 on page 83), we see in this test a slightly higher swapping rate. This means that the CPU is not waiting as long for I/O completion and the overall throughput of the three hogmem programs is higher than when the single hogmem program is running. The swapping mechanism of Linux is able, as expected, to deal with requests of more than one user at a time.

Using multiple DASD swap devices

It is possible to use more than one swap device at a time. We measure the swapping rate again with we run one hogmem program, but in this test we use two identical ECKD DASD devices of the same swap priority. By default, swap devices are set online (swapon) with different priorities. With different priorities set, they are used one after the other. In this test one hogmem program is running while two different ECKD DASD devices are prepared for swap. It is important to use ECKD DASD from different ranks of the storage server. See Example 7-8.

Example 7-8 Memory when one process swaps using two ECKD DASD

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval    <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
18:45:11   53.8   97.6    4696    2.9   938 1152  4628  4620
18:46:10   53.8   98.1    4696    2.9  2367 2154  10921 8616
18:47:10   53.8   97.2    4696    2.9  2449 2196  11188 8783
18:48:10   53.8   97.8    4696    2.9  2503 2176  11682 8705
18:49:10   53.8   97.8    4696    2.9  2533 2345  11536 9380
18:50:11   53.8   98.1    4696    2.9  2451 2150  11152 8598
18:51:11   53.8   97.1    4696    2.9  2336 2085  10812 8340
18:52:11   53.8   97.8    4696    2.9  2519 2237  11403 8947
18:53:11   53.8   98.0    4696    2.9  2402 2128  11045 8514
18:54:11   53.8   98.0    4696    2.9  2329 2143  10768 8571
```

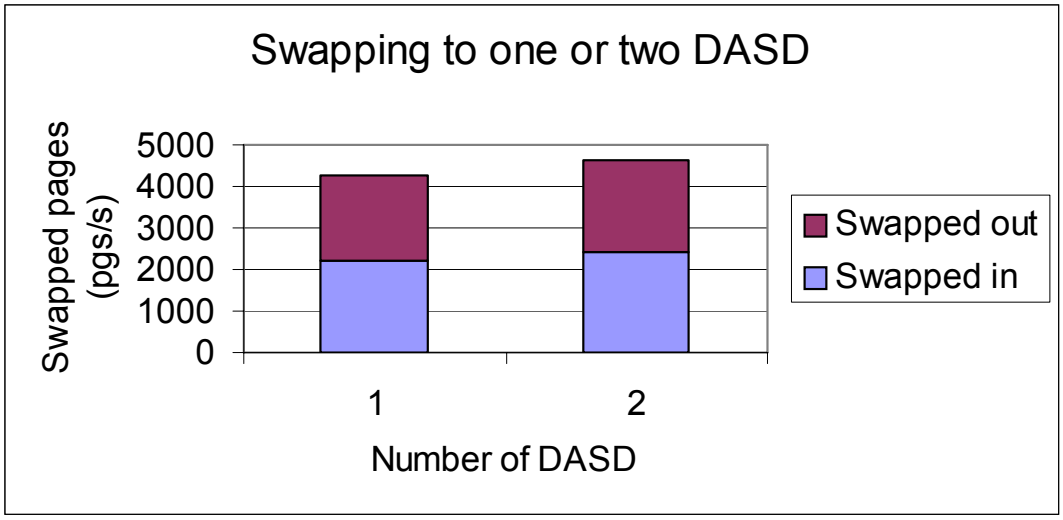


Figure 7-2 Swap rate when the hogmem program is swapping using one or two ECKD DASD

Compared to the single hogmem program running (Example 7-5 on page 83), we see in this test a higher swapping rate. This means that the CPU does not wait as long for I/O completion. The overall throughput of the hogmem program is higher when writing to two disks with the same swap priority. The swapping mechanism of Linux is able to deal with several swapping devices at the same time.

Because of time constrains in access to this measurement environment we were not able to test the two ECKD DASD environments with multiple hogmen programs running. Most likely with more hogmem programs running parallel, there would be more benefit from having multiple ECKD DASD swap devices available.

We decided to use the ECKD DASD in this test because it is widely used in the field. We expect similar results as though we ran other multiple hogmem program tests or multiple disk tests with other physical devices like FBA DISK or SCSI disk attached over FCP.

7.3.2 Swapping with DIAGNOSE access method to DASD

The Linux DASD driver uses the ECKD and FBA disciplines. In the past, the DIAGNOSE discipline had some bugs, and its use was not encouraged. The kernels shipped by SUSE and Red Hat, therefore, do not have the DIAGNOSE access method built into the kernel. This has changed over the years. Recent distributions come with the required dasd_diag_mod module and are able to use DIAGNOSE to access DASD. The latest distributions offer the DIAGNOSE option for 64-bit (z/VM 5.2 or later is required).

The DIAGNOSE access method of the Linux driver uses a high-level block I/O protocol (DIAG 250) to have CP perform the actual I/O operations when necessary. The measurements show the benefit of this protocol over the defaults chosen by the driver.

Enable DIAGNOSE I/O for the 3390 DASD

The process involves:

1. The disk has to be initially formatted under control of the ECKD discipline of the DASD driver. Before starting the Linux system, this disk was formatted with CMS and RESERVEd. Linux commands like **dasdfmt** and **mkswap** work also here.
2. Change to directory /sys/bus/ccw/devices/<addr>.
3. The command **echo 0 > online** instructs the DASD driver to stop using the device.
4. Ensure that the DIAGNOSE discipline module (dasd_diag_mod.o) is loaded.
5. The command **echo 1 > use_diag** enables the DIAGNOSE access method.
6. The command **echo 1 > online** instructs the DASD driver to use the device again.

The DASD is used by DIAGNOSE access method from this point in time.

Swapping with DIAGNOSE access method

As in the previous experiments, we use one hogmem program to drive the test. We allocate 128 MB in a 64 MB z/VM Linux guest. See Example 7-9.

Example 7-9 Memory when one hogmen program is swapping using DIAGNOSE method to DASD

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval   <--- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed   In  Out  Read Write
17:09:10  53.8    93.8    2348    5.8   2074 2528 10234 10117
17:10:10  53.8    98.1    2348    5.8   2130 1955 10402  7823
17:11:10  53.8    98.1    2348    5.8   2379 2172 11235  8690
17:12:11  53.8    97.9    2348    5.8   2498 2468 12190  9876
17:13:10  53.8    97.4    2348    5.8   2387 2187 11376  8752
17:14:11  53.8    97.8    2348    5.8   2146 2114 10222  8458
17:15:10  53.8    97.6    2348    5.9   2335 2104 11168  8436
17:16:11  53.8    97.7    2348    5.8   2465 2298 14108  9592
17:17:10  53.8    97.8    2348    5.8   2150 2084 10379  8339
17:18:11  53.8    97.5    2348    5.8   2132 2029 10325  8119
17:19:10  53.8    98.6    2348    5.8   2225 2091 10453  8365
```

In Example 7-9 on page 86 we see very similar values for swapped-in and swapped-out pages compared to the ECKD DASD measurement (Example 7-5 on page 83).

Example 7-10 CPU when one hogmen program is swapping using DIAGNOSE discipline to DASD

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
17:08:09    0.3  0.1     0.1      0  99.7      0
17:09:10    9.7  4.1     4.7   82.4   7.7    0.2
17:10:10    8.2  3.2     4.0   91.7     0    0.2
17:11:10    9.1  3.7     4.4   90.7     0    0.2
17:12:11    9.9  3.9     4.9   89.9     0    0.2
17:13:10    9.1  3.7     4.4   90.7     0    0.2
17:14:11    8.4  3.4     4.1   91.4     0    0.1
17:15:10    9.6  3.7     4.7   90.2     0    0.2
17:16:11   25.5  3.7    10.1   73.8     0    0.7
17:17:10    8.6  3.4     4.2   91.3     0    0.2
17:18:11    8.5  3.3     4.2   91.3     0    0.1
17:19:10    8.6  3.4     4.2   91.2     0    0.2
17:20:10    9.1  3.5     4.5   90.7     0    0.2
```

In Example 7-10 we observe a slight decrease in time spent waiting for I/O and a slight increase in user time. This means that the throughput of a user program using the DIAGNOSE access method to DASD is also slightly increased. Not much time is reported waiting for involuntary wait (stolen).

In the time between 17:15 and 17:16 in Linux we run a service (creating a peak) that is not related to the swap measurement. It is a service using a slow priority (low nice value set) so that the swapping itself should not be degraded.

7.4 Swapping to VDISK

Many people today suggest using z/VM Virtual Disk (VDISK) as the swap device for Linux virtual machines. A VDISK is presented to the virtual machine as a emulated fixed block architecture (FBA) DASD device (a virtual device of type 9336). Under the covers, the VDISK is an address space that lives in the z/VM main memory. The virtual machine issues I/O instructions against the device, and CP implements this by moving data between the virtual machine's primary address space and the VDISK address space.

7.4.1 Swapping with FBA discipline to VDISK

This test uses a VDISK that is accessed using FBA discipline. As with the previous experiments, we use one hogmem program to drive the test. We allocate 128 MB in a 64 MB z/VM Linux guest.

Example 7-11 Memory at swapping using FBA VDISK

```
            <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval     <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time       M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
17:29:11    53.8    97.8     488.3   27.7 1139 1448  5400   5801
17:30:11    53.8    96.2     488.3   27.7 4166 3888 19079  15558
17:31:10    53.8    97.6     488.3   27.7 4222 3867 19075  15468
17:32:11    53.8      98     488.3   27.7 4324 4016 19687  16065
```

```
17:33:10   53.8    97.7     488.3   27.7 4248 3882 18738  15526
17:34:11   53.8    97.9     488.3   27.7 3472 3365 14663  13461
17:35:11   53.8    97.9     488.3   27.7 3626 3416 15288  13665
17:36:11   53.8      98     488.3   27.7 3570 3256 14933  13023
17:37:11   53.8    97.7     488.3   27.7 3598 3441 15039  13762
17:38:11   53.8      98     488.3   27.7 3696 3408 15347  13632
17:39:11   53.8    98.1     488.3   27.7 3573 3432 15044  13730
```

Compared to the other experiments, we see a strong increase in the number of pages swapped-in and swapped-out. This means that page faults can be resolved faster, which leads to better performance of the running program. Here performance translates into throughput and transaction rate. The swap rate is close to double the amount seen in the experiments using DASD devices. See Example 7-12.

Example 7-12 CPU utilization at swapping using FBA VDISK

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
17:29:11   5.5   2.3     2.7   24.4  64.9    5.2
17:30:11  14.5   5.2     7.7   68.7    0    16.8
17:31:10  14.2   5.2     7.5   69.3    0    16.5
17:32:11  14.7   5.3     7.9   68.1    0    17.2
17:33:10  14.2   5.2     7.5     69    0    16.8
17:34:11  11.8   4.3     6.3   74.5    0    13.7
17:35:11  12.2   4.5     6.5   73.6    0    14.2
17:36:11  11.9   4.4     6.3   74.5    0    13.7
17:37:11    12   4.5     6.3     74    0      14
17:38:11  12.1   4.6     6.3   73.7    0    14.3
17:39:11  12.1   4.4     6.4   73.9    0      14
```

We get completely different results for the CPU utilization compared to the previous measurements (Example 7-6 on page 83) using a real device. I/O wait time is down compared to the previous measurements. Unfortunately, this increases the user time only to a small degree. Most of the saved I/O wait time is now reported as stolen.

7.4.2 Swapping with DIAGNOSE access method to VDISK

We run this next test swapping to a VDISK using the DIAGNOSE access method. As with the previous experiments we use one hogmem program to drive the test. We allocate 128 MB in a 64 MB z/VM Linux guest.

Example 7-13 Memory at swapping using DIAGNOSE VDISK

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval    <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
17:49:11  53.8    97.9   488.3   27.7  1453 1831  6716   7335
17:50:10  53.8    97.7   488.3   27.7  5211 4790 23583  19159
17:51:10  53.8    97.8   488.3   27.7  5841 5518 26231  22071
17:52:11  53.8    97.7   488.3   27.7  6045 5599 26971  22395
17:53:11  53.8    97.5   488.3     28  5941 5585 28257  22356
17:54:10  53.8    96.6   488.3     28  5458 5138 25464  20555
17:55:11  53.8    97.5   488.3     28  5804 5374 26747  21499
17:56:11  53.8    97.8   488.3     28  6046 5534 27217  22138
17:57:10  53.8    97.8   488.3     28  5868 5532 26620  22129
```

```
17:58:11  53.8      98   488.3     28  5824 5417 25866  21666
17:59:11  53.8    97.2   488.3   27.9  5116 4818 22223  19274
```

The swap rate is higher when we compare the results to the swap rate accessing the same VDISK using the FBA method. Also, read and write block I/O are significantly higher than seen in the FBA VDISK test. See Example 7-14.

Example 7-14 CPU utilization at swapping using DIAGNOSE VDISK

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
17:49:11      7    3     3.4   22.7  66.5   3.8
17:50:10   17.7  6.4     9.4   70.4    0   11.9
17:51:10   19.7  7.2    10.4   66.3    0   13.9
17:52:11   20.2  7.4    10.7   65.5    0   14.3
17:53:11   21.5  7.8    11.4     64    0   14.5
17:54:10   18.7  6.7    10.1   68.4    0   12.9
17:55:11   19.6  7.1    10.4   66.8    0   13.6
17:56:11   20.3  7.4    10.9   65.5    0   14.2
17:57:10   19.7  7.2    10.5   66.3    0     14
17:58:11   19.5  7.2    10.3   66.4    0   14.1
```

We get similar results for the CPU utilization compared to the previous measurement (Example 7-12 on page 88) using VDISK. I/O wait time is down compared to the measurement using the FBA discipline accessing the same VDISK. The time attributed to stolen is also slightly down. The user time is up, which translates into better performance (throughput, transaction rate) of the application.

7.4.3 The advantages of a VDISK swap device

The advantages of VDISK are that a very large swap area can be defined with very little expense. The VDISK is not allocated until the Linux server attempts to swap.

Enabling an FBA VDISK

Because a VDISK appears as an FBA device to Linux, the DASD driver uses the FBA discipline by default for both Novell SUSE and Red Hat distributions.

Just as with the ECKD swap device, the VDISK must be initialized with the **mkswap command** before the **swapon** command can use the device. Because the contents of the VDISK are not preserved after logoff of the virtual machine (the data on VDISK is volatile), the **mkswap** command must be issued each time you start the virtual machine.

There are at a few different ways to do this:

- Modify the init scripts in your Linux system to run the **mkswap** command early in the boot process. The disk can then be picked up automatically by the **swapon** command when Linux processes the /etc/fstab file. The **swapon** command can be issued manually if required.
- First IPL CMS in the Linux guest, and then use CMS tools to initialize the VDISK. This enables Linux to see the VDISK as a swap device just as though a **mkswap** command was already issued. If you also need to use CMS to couple virtual channel-to-channels (CTCs), this may be a good option.
- Use some other virtual machine to link to all the VDISKs so that CP retains them after the Linux guest is logged off. You still need a process to initialize the disks after a z/VM IPL.

This approach is probably not very attractive because it causes CP to retain more VDISKs than you need and puts an additional burden on the paging subsystem.

The choice for initializing the disk in Linux or in CMS depends on the skills available and whether you are willing to change things in Linux. Neither the Novell SUSE nor the Red Hat installers currently use VDISK as the swap device, but if you prepare the disk on CMS in advance, Linux picks it up automatically.

Note: Writing to the raw VDISK in CMS requires serious programming. The RSRVDISK EXEC (shown in 7.10, “Initializing a VDISK using CMS tools” on page 100) is an example of a script to initialize a VDISK.

Effect of parallel swapping to VDISK

Some recommendations for swapping involve the use of multiple swap partitions. When multiple swap partitions with the same priority are used, Linux effectively spreads the I/O over multiple disks and can achieve a higher I/O rate. When using VDISKs, however, this does not apply. There is very little queuing for the VDISK, and any further increase of swap space is unlikely to provide any benefit.

7.5 Swapping with FCP protocol to Linux attached SCSI disk

Linux on System z and z/VM running on System z hardware support the Small Computer System Interface (SCSI) disks attached to FICON® Express running in FCP mode. As in the previous experiments, we use one hogmem program to drive the test. We allocate 128 MB in a 64 MB z/VM Linux guest.

Example 7-15 Memory when swapping to a SCSI disk over FCP

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval    <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time    M_Total %MUsed  S_Total %SUsed   In   Out  Read Write
18:07:11  53.8    97.1   3815    3.5  839.6  1188  4483  4764
18:08:11  53.8    98.0   3815    3.5   2789  2690 12655 10760
18:09:10  53.8    98.2   3815    3.5   2681  2509 12080 10036
18:10:10  53.8    98.1   3815    3.5   2949  2823 13846 11293
18:11:10  53.8    96.7   3815    3.5   2966  2782 13720 11128
18:12:10  53.8    98.0   3815    3.5   2877  2592 13071 10370
18:13:10  53.8    97.8   3815    3.5   2842  2666 13148 10666
18:14:11  53.8    97.9   3815    3.5   2876  2707 13038 10829
18:15:11  53.8    97.4   3815    3.5   2816  2656 12871 10628
18:16:10  53.8    97.7   3815    3.5   2844  2688 13117 10751
18:17:10  53.8    97.3   3815    3.5   2759  2647 12593 10589
```

In Example 7-15 we see higher rates for swapped-in and swapped-out pages compared to both DASD measurements, but less than swapping to VDISK.

Example 7-16 CPU utilization swapping to SCSI disk over FCP

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
18:08:11    11.1  4.2    5.0   88.9    0     0.1
18:09:10    10.6  4.0    4.8   89.3    0     0.1
18:10:10    12.0  4.4    5.6   87.9    0     0.1
18:11:10    11.9  4.4    5.6   88.0    0     0.1
```

```
18:12:10    11.1  4.3    5.0   88.8    0     0.1
18:13:10    11.4  4.2    5.3   88.5    0     0.1
18:14:11    11.2  4.3    5.1   88.7    0     0.1
18:15:11    11.1  4.2    5.1   88.8    0     0.1
18:16:10    11.4  4.2    5.4   88.5    0     0.1
18:17:10    11.0  4.1    5.1   88.9    0     0.1
```

The I/O wait time in Example 7-16 on page 90 is less than that seen running the DASD tests. More is attributed to user time, which improves the throughput and transaction rate of applications. More stolen time is reported than when running the DASD tests. The SCSI characteristics are slightly better than DASD. Not much time is seen for involuntary wait (stolen).

7.6 Swapping to DCSS

We found improved support for DCSS in recent distributions. From Novell SUSE SLES9 SP2 and Red Hat RHEL4 Update4 it is possible to define swap space in DCSS. There are some advantages to using a DCSS for swapping:

- This option offers fast write into z/VM's storage and swap caching when the Linux guest is memory constrained but z/VM is not.
- It allows you to shrink guest virtual memory size while maintaining acceptable performance for peak workloads.
- This solution requires no channel programs like VDISK access. Data transfers done by z/VM memory management only.
- Low SIE break-rate is seen.

Swapping to DCSS is the fastest known method. As with VDISK, the solution requires memory. But lack of memory is the reason for swapping. So it could be used as preferably as a small fast swap device in peak situations. The DCSS swap device should be the first in a cascade of swap devices, where the following could be bigger and slower (real disk). The swapping to DCSS adds complexity. We did not measure the swapping to DCSS.

7.7 Comparison of swap rates of tested swap options

Here we compare all the measurements of swap rates and the CPU consumption of the various tested swap options.

Comparison of swap rates of the various options

Swapped-in and swapped-out rates are added to an overall swap rate. The swap rate is a performance measure of how fast pages can be transferred to the swap device or back from it. The numbers show the Linux point of view, so keep in mind that this experiment is running on top of the z/VM virtualization layer. See Figure 7-3.

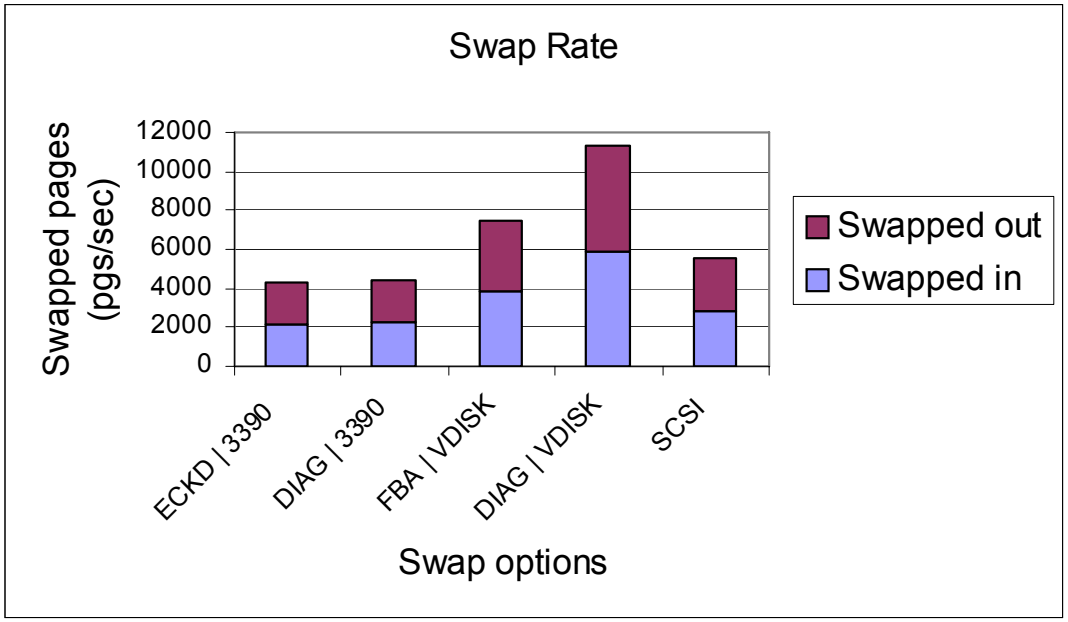


Figure 7-3 Comparison of swap options

We do not observe much of a difference in the swapping rate to a DASD using ECKD discipline or the DIAGNOSE access method. A better performance has the SCSI disk. Even faster is VDISK using FBA discipline. The fastest tested option was VDISK using the DIAGNOSE access method. The result is as expected since VDISK resides in memory.

Whether VDISK is the correct choice depends not only of the performance but also of the scenario. If the environment is short on memory, VDISK is not the correct choice. If you need to swap to real disks, SCSI is the most effective option.

The next chart (Figure 7-4) has a comparison of the total CPU and the details of CPU consumption.

Note: We ran the test over a constant time, not to finish a specified amount of work. If we had a fixed amount of work to run, it would have finished earlier if the request for page allocation could have been satisfied faster. The bars would then not only be of a different height, but also would be wider on the x-axis over time.

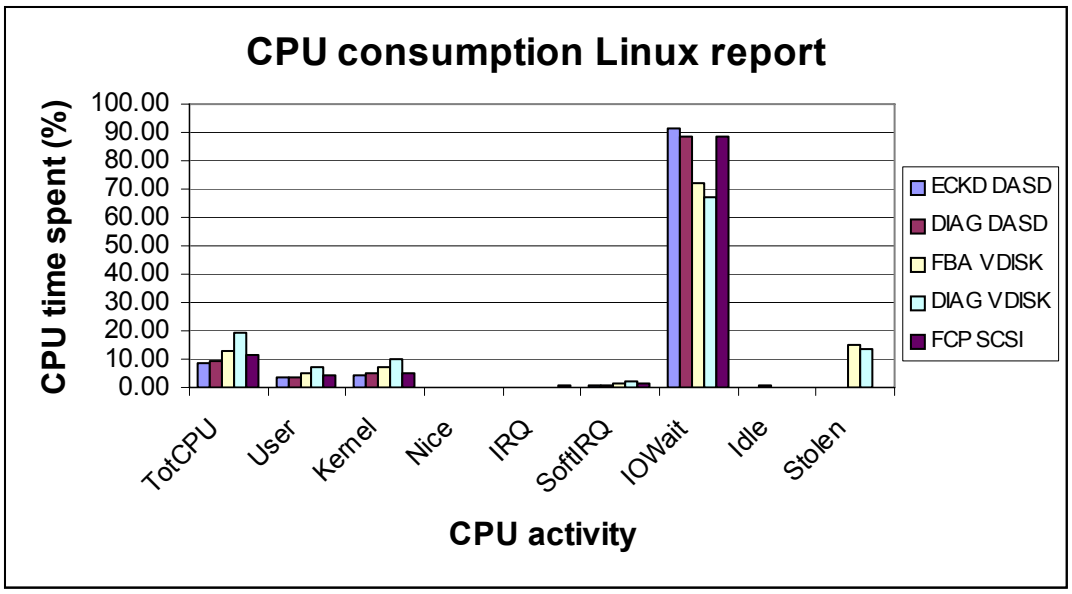


Figure 7-4 Linux CPU consumption report for swap options

Most of the time is reported to IOWait, which is expected. I/O that is really going out to devices and that cannot be satisfied by any caches can be expected to be magnitudes slower than other operations living in memory. We see lower values for the VDISK tests counted to IOWait. Stolen time is reported for VDISK only. The highest Total CPU (TotCPU)) and kernel CPU usage is reported for VDISK. For the application performance, the user CPU is the most important part, which shows highest values for combinations DIAGNOSE VDISK and FBA VDISK.

Remember that the results are not in ratio to the swapped pages, so fast options swap more in am amount of time and need more CPU time. This explains the high TotCPU values for the VDISK options.

Comparison of the costs as seen by z/VM

We use the Performance Toolkit and PAF to illustrate the CPU costs as seen by z/VM.

The bars in Figure 7-5 show the measurements in order from left to right:

- ECKD DASD
- DIAG DASD
- FBA VDISK
- DIAG VDISK
- SCSI DISK

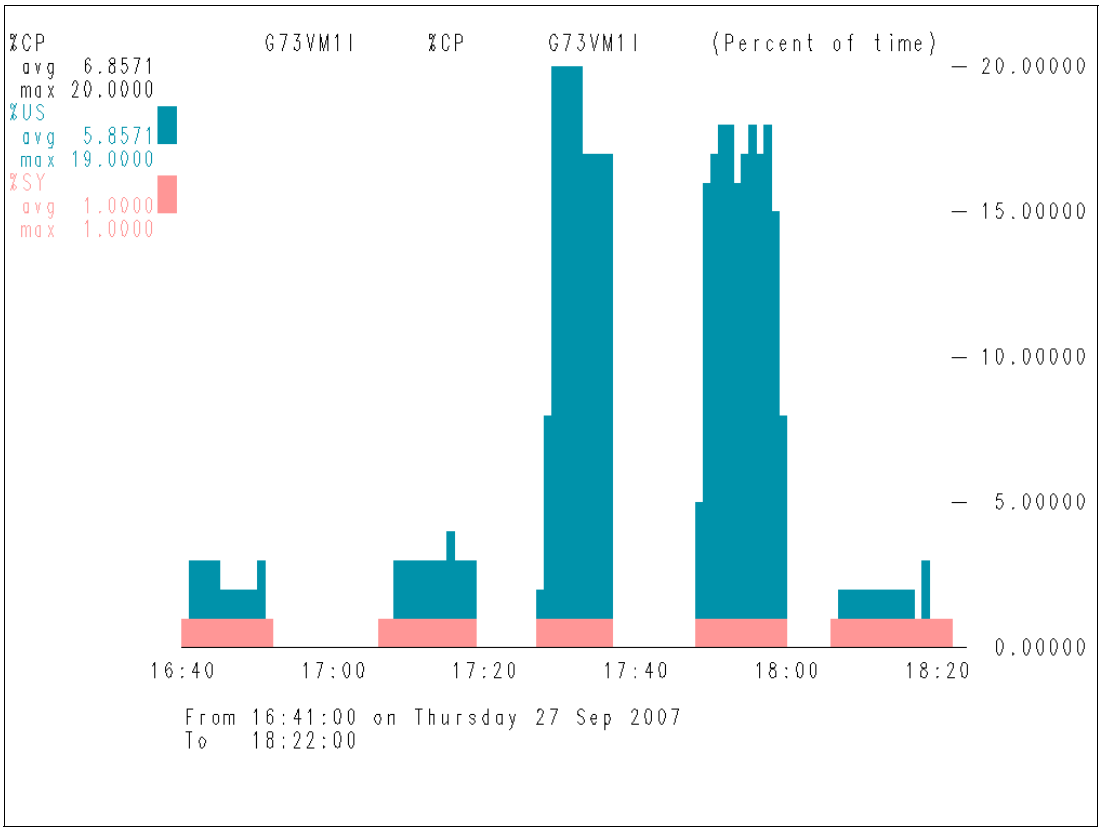


Figure 7-5 CPU costs as reported by z/VM

The costs counted on System CPU are constant while running the different experiments. The VDISK experiments (bar 3 and 4) have far higher CP supervisor user CPU values than all of the other options considered. But VDISK tests get through a given work unit faster, and here the experiment runs a specified elapsed time, rather than finishing a specified amount of work. This proves that faster VDISK swapping needs relatively more overall CPU at the time that it is happening. Although the SCSI was faster than the ECKD DASD and DIAGNOSE DASD swapping, it is reported to need fewer USER CPU resources.

7.8 Recommendations for swapping

The worst-case scenario is when, for whatever reason, applications need more memory than usual, and that request cannot be satisfied. This could be caused by newly introduced bad program code (memory leak) or the application starting a large number of threads. As a result, the Linux kernel starts to kill processes using the OOM-killer Linux kernel function. We see this as the worst case because it would lead to abnormal application ends and potential data loss. Even if we swap to slow devices and bring down the performance, it is definitely better than data loss and abnormal ending programs.

Plan swapping in advance. If swapping is to be expected as an exception, the swap devices should be of high performance (that is, if running under z/VM to define VDISK as the swap device with the highest priority). VDISK resides in main memory. So, if main memory is a constrained resource in the environment, we do not recommend the usage of VDISK. Swapping on a regular base is observed at higher resource overcommit rates, as often seen in low-utilized, consolidated environments.

- Size the Linux virtual machine to reduce the amount of Linux swapping.

  The optimum virtual machine size is a trade-off between reducing overall z/VM memory usage and reducing swapping in a Linux guest. As discussed in 2.3, “Sizing considerations for z/VM Linux guests” on page 11, reduce the virtual machine size of the Linux guest to the point where swapping begins under normal load, and then add an additional amount to minimize Linux swapping.

- The amount of swap space to allocate depends on the memory requirements of your Linux guest.

  The suggestion that swap space should be twice the memory size of a Linux machine should not apply to a z/VM Linux guest. If a Linux guest uses this much swap space, it probably indicates that a larger virtual machine size should be allocated to the guest.

- Do not enable MDC on Linux swap minidisks.

  The read ratio is not high enough to overcome the write overhead.

- Swapping to VDISK with the DIAGNOSE access method is faster than swapping to DASD or SCSI disk.

  In addition, when using a VDISK swap device, your z/VM performance management product can report swapping by a Linux guest.

- Consider multiple swap devices rather than a single, large VDISK swap device.

  Using multiple swap devices with different priorities can alleviate stress on the VM paging system when compared to a single, large VDISK. As discussed in 7.4.1, “Swapping with FBA discipline to VDISK” on page 87, a VDISK combined with a DASD swap device can provide a small, fast swap option (the VDISK) with spillover to a larger, slower DASD swap device.

- Prefer swap devices over swap files.

  Linux on System z is able to use swap files instead of swap partitions or swap disks. The swap files introduce extra effort to handle the file system.

7.8.1 Cascaded swap devices

There is a good case for using multiple swap disk devices with different priorities. If multiple swap devices with different priority are available, Linux attempts to fill the one with the highest priority before using the next device. The algorithms for allocating pages on swap devices in Linux cause the active area to *sweep* over the device, which reduces seek times on the device and increases the ability of Linux to build long I/O chains.

This means that over time the entire swap device has been referenced. In the case of VDISK, this means that CP has to provide memory for the entire VDISK, even though the Linux virtual machine might only need a small portion of the VDISK at any given time. If memory contention is high enough, the contents of the VDISK are paged out by CP and must be brought back in when Linux next references that part of the VDISK.

If we give the Linux virtual machine two smaller VDISKs instead of one big VDISK and use them as swap devices with different priorities, the *footprint* is effectively reduced by 50%.

We recommend using a smaller VDISK with higher priority and a bigger disk (DASD, SCSI) with a lower priority in cases where swapping is expected in peak situations. In environments where paging is always expected, the usage of VDISK has no advantage, since it reduces the amount of available memory.

7.8.2 Impact of page-cluster value

The page-cluster value determines how many additional pages Linux will swap in on a page fault (under the assumption that the process will want the next pages as well). A value of 1 translates in this context to 2 (2 power 1). The default value for the page-cluster size is 3, which makes that the unit seen as 8 pages (2 power 3). Page-clustering is controlled by the /proc/sys/vm/page-cluster pseudo-variable.

In Example 7-18, we illustrate the effect of page-clustering. The program for the test is again hogmem (7.9, “Program text for hogmem test” on page 99). Tests starts at 16:29. Before starting the test, the page-cluster size is set to one:

```
# echo 1 > /proc/sys/vm/page-cluster
```

Between 16:34:33 and 16:35:33, the page-cluster value is set to three (default value on SLES10 SP1):

```
# echo 3 > /proc/sys/vm/page-cluster
```

Swap activity as reported by **vmstat** showed no significant change during this period for swapped-in and swapped-out pages, but it shows the changes in the I/O values for blocks in and blocks out, and more interrupts and context switches are decreased.

Example 7-17 Changed page-cluster value observed by vmstat

```
g73vm1:~ # vmstat 60
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  2 139604   1156     84   1584 5955 5599  6638  5599  855 1563  2  4  0 93  0
 0  1 139604   1712     48    384 6306 6554  6851  6559  877 1652  3  5  0 93  0
 0  2 139604   2168     48   1368 5618 5491  5992  5492  788 1472  2  4  0 94  0
 0  1 139604   1608     44    844 5868 5969  6198  5969  814 1534  2  4  0 94  0
 0  2 139604   1096     44    608 5960 5525  6451  5525  842 1557  2  4  0 93  0
 0  2 139604   1340     60   4344 6187 6124  6640  6124  870 1614  2  4  0 93  0
 0  3 139604   1808     48    816 7234 7122  7633  7122  498  868  3  4  0 93  0
 0  1 139604   1360     48    940 8586 7792  9086  7792  396  640  3  4  0 92  0
 0  1 139604   1536     48     96 8186 8112  8622  8112  384  610  3  4  0 93  0
 0  1 139604   1420     48    304 8602 8033  9105  8033  380  641  3  4  0 92  0
 0  3 139604   1312     48    904 7851 7373  8308  7373  372  594  3  4  0 93  0
 0  1 139604   1104     48    796 8427 7762  8988  7762  394  636  3  4  0 93  0
 0  2 139604   1068     48    536 8261 7438  8714  7438  420  622  3  4  0 93  0
```

We are interested in comparing the Linux point of view to with the performance toolkit results.

Example 7-18 Changed page-cluster value observed by performance toolkit

```
 FCX162      CPU 2084  SER F8OCA  Interval 01:53:46 - 16:44:33    Perf. Monitor

 Resource Usage Log for User G73VM1

           <----- CPU Load -----> <------ Virtual IO/s ------>
 Interval       <-Seconds->   T/V
```

End Time	%CPU	TCPU	VCPU	Ratio	Total	**DASD**	Avoid	Diag98	UR	Pg/s	User Status
>>Mean>>	...	....	....	...	...	...	...	...	...	...	---,---,---
16:29:33	8.80	5.279	4.352	1.2	828	**828**	.0	.0	.0	.0	EME,CLO,DIS
16:30:33	9.05	5.432	4.469	1.2	850	**850**	.0	.0	.0	.0	EME,CLO,DIS
16:31:33	8.22	4.933	4.059	1.2	781	**781**	.0	.0	.0	.0	EME,CLO,DIS
16:32:33	7.83	4.700	3.868	1.2	751	**751**	.0	.0	.0	.0	EME,CLO,DIS
16:33:33	8.38	5.025	4.118	1.2	790	**790**	.0	.0	.0	.0	EME,CLO,DIS
16:34:33	8.85	5.308	4.356	1.2	836	**836**	.0	.0	.0	.0	EME,CLO,DIS
16:35:33	8.16	4.896	4.150	1.2	562	**562**	.0	.0	.0	.0	EME,CLO,DIS
16:36:33	8.50	5.099	4.467	1.1	345	**345**	.0	.0	.0	.0	EME,CLO,DIS
16:37:33	8.68	5.208	4.568	1.1	342	**342**	.0	.0	.0	.0	EME,CLO,DIS
16:38:33	8.99	5.396	4.730	1.1	359	**359**	.0	.0	.0	.0	EME,CLO,DIS
16:39:33	8.25	4.949	4.340	1.1	340	**340**	.0	.0	.0	.0	EME,CLO,DIS
16:40:33	8.59	5.151	4.514	1.1	354	**354**	.0	.0	.0	.0	EME,CLO,DIS
16:41:33	8.69	5.212	4.566	1.1	352	**352**	.0	.0	.0	.0	EME,CLO,DIS
16:42:33	8.56	5.134	4.490	1.1	363	**363**	.0	.0	.0	.0	EME,CLO,DIS

We note in Example 7-18 on page 96:

- A higher DASD I/O rate in the interval 16:29:33 to 16:34:33. The page-cluster value in this interval is set to 1.
- The DASD I/O rate decreases dramatically at 16:34:33/16:35:33. Beginning in this interval, the page-cluster value is returned to its default of 3.

This test shows that a small page_cluster value causes many short I/Os.

In Figure 7-6 we show the real DASD I/O measured by the performance toolkit. This graphic was produced using PAF. We see here that the number of I/Os decreases after /proc/sys/vm/page-cluster was set back to the default of 3.

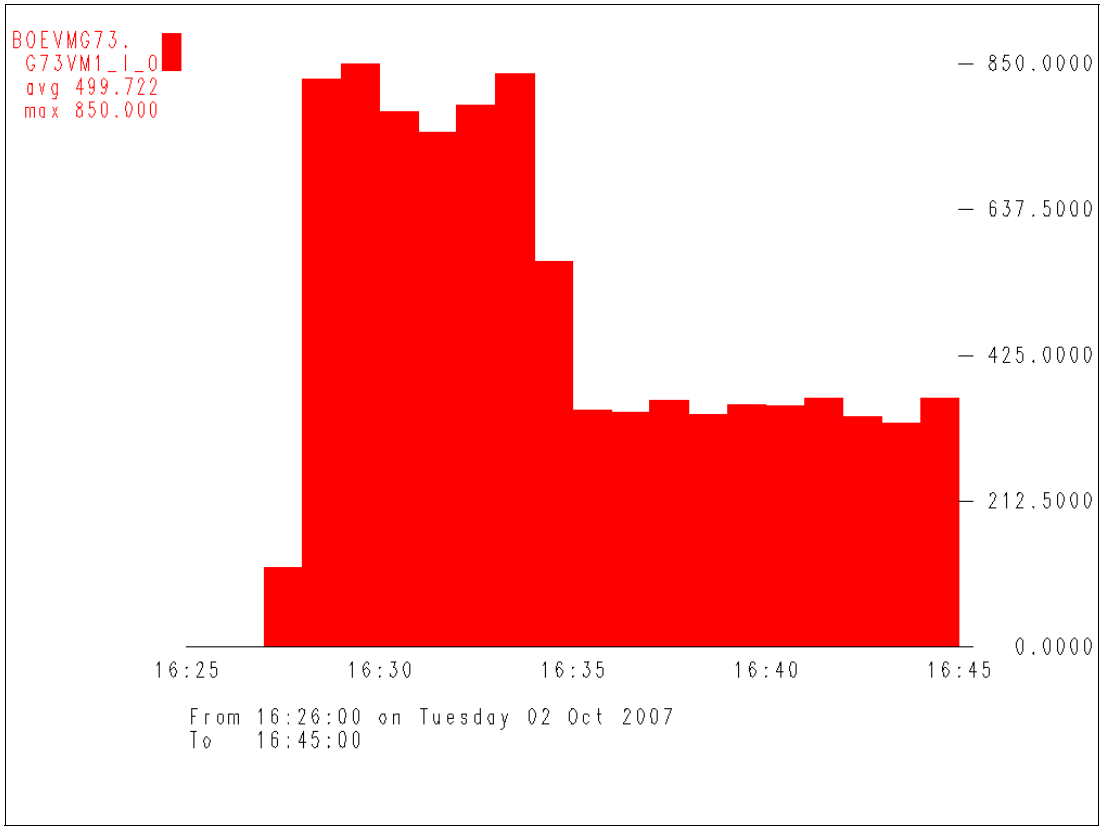


Figure 7-6 Real DASD I/O for swap disk DASD 4443

Using a page_cluster size of 1 speeds up swapping with the hogmem test program but increases the number of interrupts and context switches in Linux. It increases the virtual and real DASD I/O reported in z/VM.

Note: The test results differ from our expectations and former recommendations to use a page-cluster size of 1. We found the reason in the characteristic of the hogmem test program. It touches the pages walking linearly through the memory. So bigger units of page clusters that where swapped in at once include the pages needed in the next moment. If the test program would touch pages randomized, the swapping of eight pages at once would bring in many pages that where not needed.

7.9 Program text for hogmem test

With hogmem, it is easy to run processes that allocate and use virtual memory. We use this to compare efficiency of the different swap devices in Linux. The program is shown in Example 7-19.

Example 7-19 Listing of hogmem.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
#include <signal.h>
#include <time.h>
#include <sys/times.h>

#define MB (1024 * 1024)

#ifndef CLK_TCK
# define CLK_TCK CLOCKS_PER_SEC
#endif

int nr, intsize, i, t;
clock_t st;
struct tms dummy;

void intr(int intnum)
{
    clock_t et = times(&dummy);

    printf("\nMemory speed: %.2f MB/sec\n", (2 * t * CLK_TCK * nr + (double) i * CLK_TCK *
intsize / MB) / (et - st));
    exit(EXIT_SUCCESS);
}

int main(int argc, char **argv)
{
    int max, nr_times, *area, c;

    setbuf(stdout, 0);
    signal(SIGINT, intr);
    signal(SIGTERM, intr);
    intsize = sizeof(int);
    if (argc < 2 || argc > 3) {
        fprintf(stderr, "Usage: hogmem <MB> [times]\n");
        exit(EXIT_FAILURE);
    }
    nr = atoi(argv[1]);
    if (argc == 3)
        nr_times = atoi(argv[2]);
    else
        nr_times = INT_MAX;
    area = malloc(nr * MB);
    max = nr * MB / intsize;
    st = times(&dummy);
    for (c = 0; c < nr_times; c++)
    {
        for (i = 0; i < max; i++)
            area[i]++;
        t++;
```

```
        putchar('.');
    }
    i = 0;
    intr(0);
    /* notreached */
    exit(EXIT_SUCCESS);
}
```

The program also reports a *memory bandwidth* in MBps, but it should be clear that this has little value for Linux in a virtual machine.

7.10 Initializing a VDISK using CMS tools

The RSRVDISK EXEC, shown in Example 7-20, can be used to initialize a VDISK in CMS before starting Linux. This is only sample code to illustrate what needs to be done. You should customize this script and call it from PROFILE EXEC.

Example 7-20 RSRVDISK EXEC: make a VDISK into CMS RESERVEd

```
/* RSRVDISK EXEC     Format and Reserve a fresh VDISK               */
arg cuu .
if cuu = '' then signal usage

'PIPE (end \)',
  '\ command QUERY DISK',
  '| drop',
  '| spec 13 1',                      /* All used filemodes         */
  '| strliteral /ABCDEFGHIJKLMNOPQRSTUVWXYZ/',
  '| fblock 1',
  '| sort',
  '| unique single',
  '| take 1',
  '| append strliteral /*/',          /* A default                  */
  '| var fm'

if fm = '*' then call emsg 36, 'No free filemode found'
cuu = right(cuu,4,0)

queue '1'
queue 'SW'cuu
'FORMAT' cuu fm '( BLK 4K'
queue '1'
'RESERVE' userid() 'SWAP'cuu fm
'RELEASE' fm

return

emsg:
  parse arg rc, txt
  say txt
  if rc ¬= 0 then exit rc
return

usage: say 'SWAPDISK cuu'
```

After running the program, start Linux and initialize the disk with the **mkswap** command. This does not undo the CMS formatting, but only causes Linux to write some blocks in the *payload*

of the disk (the single CMS file created on the disk). Then, without running the **swapon** command, shut down the Linux guest and IPL CMS again. Example 7-21 shows how to copy the modified blocks.

Example 7-21 Copying the modified blocks from the RESERVEd disk

```
list * * K (date
FILENAME FILETYPE FM FORMAT LRECL       RECS     BLOCKS   DATE     TIME
RMHTUX02 SWAP0207 K6 F       4096      16360      16360  2/21/03  5:21:36
Ready; T=0.01/0.01 05:28:46
pipe diskrandom rmhtux02 swap0207 k number 1-16360 | strip trailing 00 | locate
11 | > sample swap0207 a | chop 10 | cons
         1
Ready; T=1.39/1.59 05:30:10
```

The PIPE command in Example 7-22 creates a small file on the A disk to hold the modified blocks (it shows that only a single block was modified). This file can be used to prepare a fresh VDISK again.

Example 7-22 Prepare a fresh VDISK

```
list * * c
RMHTUX02 SWAP0207 C6
Ready; T=0.01/0.01 05:47:57
pipe < sample swap0207 | pad 4106 00 | fileupdate rmhtux02 swap0207 c6
Ready; T=0.01/0.01 05:48:18
```

To Linux, that new VDISK now looks just like the one that was prepared with the **mkswap** command before. If you take the time to study the contents of the SAMPLE SWAP0207 file, you find that it is easy to create the contents from scratch and even make it work for VDISKs of any size.

Tip: A similar process can be used if you want VDISK to hold temporary files for Linux. In this case, run the **mke2fs** command instead of the **mkswap** command against the device before you copy the payload from it. If necessary, you can also create directories (and even files) to have the disk in the correct state for when Linux boots.

8

CPU resources and the z/VM scheduler

In this chapter we cover CPU resources and the z/VM scheduler. Topics include:

- Understanding LPAR weights and options
- The CP scheduler
- Virtual machine scheduling
- CP scheduler controls
- Analysis of the SET SRM LDUBUF control
- Virtual Machine Resource Manager
- Steal time

© Copyright IBM Corp. 2003, 2008. All rights reserved.

8.1 Understanding LPAR weights and options

Two of the reasons that you may wish to configure at LPAR level are:

- To consolidate multiple slower processors onto much faster z9s
- To separate workloads through the use of LPAR mode to avoid issues with the 2 GB line, if you are still using 31-bit Linux and applications

This section clarifies some of the configuration options.

Figure 8-1 shows extracts from a logical partition (LPAR) report, produced with the VM Performance Tool Kit. Because z/VM often operates in multiple LPAR environments, understanding LPAR options and their impact on performance helps you configure your systems to meet your business requirements.

```
LPAR Data, Collected in Partition A02

Processor type and model    : 2094-710
Nr. of configured partitions:      45
Nr. of physical processors  :      18
Dispatch interval (msec)    : dynamic

Partition Nr.  Upid #Proc Weight Wait-C Cap %Load CPU %Busy %Ovhd
A0A          1    10     2     10     NO  NO    .4   0   3.3    .1
                                 10         NO         1   3.3    .1
A0B          2    ..     0           NO          0 ...   ...   ...
A0C          3    ..     0           NO          0 ...   ...   ...
A0D          4    13     1     10     NO  NO   ...   0    .4    .1
A0E          5    14     1     10     NO  NO   ...   0    .3    .0
A0F          6    15     1    DED    YES  NO   ...   0 100.0    .0
A01          7    01     4     10     NO  NO    .2   0    .6    .0
                                 10         NO         1    .7    .0
                                 10         NO         2    .7    .0
                                 10         NO         3    .7    .1
A02          8    02     4     10     NO  NO    .3   0   1.2    .0
                                 10         NO         1   2.6    .0
                                 10         NO         2   1.0    .0
                                 10         NO         3   1.2    .0
A03          9    03     2     10     NO  NO   ...   0    .0    .0
                                 10         NO         1    .0    .0
A04         10    04     6     10     NO  NO    .4   0   3.5    .1
                                 10         NO         1   3.4    .1
                                 10         NO         2    .0    .0
                                 10         NO         3    .0    .0
                                 10         NO         3    .0    .0
A28         44    40     4     10     NO  NO    .4   0   1.6    .0
                                 10         NO         1   1.6    .0
                                 10         NO         2   1.6    .1
                                 10         NO         3   1.7    .1
A29         45    41     2     10     NO  NO    .2   0   2.1    .1
                                 10         NO         1   1.9    .1

General LPAR mgmt overhead                        1.7
Overall physical load                             9.0
```

Figure 8-1 LPAR report

In Figure 8-1 on page 104, the z/VM LPAR in which the reported data is collected (A02) is shown. The overall report, which may be referenced from Appendix E, “Additional material” on page 211, shows what z/VM considers its processor utilization to be for each LPAR, as well as the total for the all partitions in the entire CPU machine. The term %Busy is the time that a physical processor is assigned to a logical one. LPARs are prioritized by weights, which are explained in 8.1.2, “Converting weights to logical processor speed” on page 106.

All of the %Busy and other processor utilizations shown in this report are in absolute numbers, meaning that these values are percentages of one processor. There is never a case of a reported percent of a percent, where one of the percents is not provided. Therefore, after the four columns for A02 are summed, the VM LPAR is shown as using 6% of a possible 400% (four processors).

The values reported for A02 and other LPARs in this example include:

- Processor type and model.
- Nr. of configured partitions - total number of partitions on the entire system.
- Nr. of physical processors - total number of physical processors installed and enabled.
- Dispatch interval (set to dynamic).
- Partition - the name given to the logical partition.
- Nr. - the number of the logical partition.
- Upid - the user partition ID for the logical partition.
- #Proc - the total number of processors defined for the partition.
- Weight - determines the share of time the partition is granted control of the processors.
- Wait completion (set to no) - This option determines whether the logical processor keeps running on the real processor until the time slice allocated is complete, even though it does not have work to execute and is waiting. If this is required, set the option to YES. If you want to hand control to other processors before your time slice is up, when you are only waiting and no longer executing, set this to NO.
- Cap (set to no) - If the option set here is yes, this indicates that CPU is provided up to the allocated maximum only. If set to no, CPU is provided up to the maximum, and beyond if it is available.
- %Load - This is the relative load of this partition, based on the time its logical processors were active, divided by the totally available processor time, and expressed as a percentage.
- CPU - the CPU identification character allocated to each processor in the partition. This starts at 0 and counts up to the total number of CPUs assigned.
- %Busy - percentage of time that the logical processor was busy, and was assigned to a real processor.
- %Ovhd - This is the percentage of elapsed time spent on the management of LPAR itself, the overhead for running and controlling the various workload partitions it is responsible for.

Some of these are discussed further in 8.1.4, “LPAR options” on page 107.

The following LPAR values are not shown in the example, but are reported by the VM performance tool kit:

- %Susp - This is the percentage of time that the logical processor was suspended and could not provide service to the guest system due to LPAR management time and contention for real processors.

- % VMLD - the processor load as seen by the VM guest, based on the processor busy time seen by the guest system and elapsed time.
- %Logid - the processor load as seen by the VM guest, based on the processor busy time seen by the guest system and the sum of wait time and processor busy time perceived by the guest VM system.
- Type - CPU type of the logical processors defined for the partition. The types listed below are seen only when using z/VM systems V5.3 and later. For earlier releases, this field always shows CP only, regardless of the actual.
- The processor type:
  - CP - Central processor
  - ICF - Internal Coupling Facility Processor
  - IFL - Integrated Facility for Linux Processor
  - ZIIP - IBM z9 Integrated Information Processor (zIIP)
  - ZAAP - IBM System z Application Assist Processor

There is a correlation between the number of logical processors defined and active and the amount of overhead involved in time slicing the physical processor between the logical processors. The more logical processors, the higher the overhead.

8.1.1 LPAR overhead

The LPAR overhead of managing all of the various partitions under its control is broken down into two values by the VM Performance Tool Kit. The term %Ovhd, given for each logical processor, is the overhead calculated as the percent of elapsed time that LPAR has spent running and controlling the partitions. In the example shown, the total overhead for all processors in all partitions is 5.7%. The other figure shown at the end of the listing is the general LPAR management overhead, which in this example is 1.7%.

Note: The IBM VM Performance Tool Kit report shows all processors types. General use central processors (CPs), Integrated Facility for Linux (IFL), Internal Coupling Facility (ICF), z9 Integrated Information Processor (zIIP), and System z Application Assist Processors (zAPP) are all shown.

From the report, we see that there are 45 partitions defined. There are 106 logical processors shared between various active guests, with 16 physical (non dedicated) processors on which to dispatch them (0–6), each with an overhead in the 5–6% range. The two dedicated processors (partition numbers 6 and 19) have no overhead in the example. However, this does not imply that the processors should be dedicated to LPARs to reduce overhead. The number of LPARS defined and the number of logical processors defined clearly relates to the total overhead costs.

Note: To reduce physical overhead, use fewer LPARs and fewer logical processors.

8.1.2 Converting weights to logical processor speed

An LPAR is granted control of processors based on time slices. Each LPAR gets time slices based on the weighting factor for the LPAR. To determine the weight of each logical processor, use the following calculation:

1. Add up all the weights for the logical processors.

In Figure 8-1 on page 104, there are 2 dedicated partitions and 43 sharing LPARs defined, 34 of which are in use, sharing 16 logical processors (0–15) based on weighting. The total weight of all the logical processors is 330.

2. Divide the weight of the *interesting LPAR* into the total.

   This is the *logical share* of the physical complex allocated to the interesting LPAR. LPAR A02 running z/VM has a weight of 10. Dividing this by the total shares (330) yields a 3% share of the sixteen shared processors.

3. Divide the number of logical processors of the LPAR into the logical share.

   This is the share of each logical processor that is directly relative to the maximum speed at which a logical processor operats if capped. Thus, 3% of 16 processors is equivalent to almost half of one processor (3x16 = 48%). Divide this between the four logical processors used by our VM system in LPAR A02, and each processor is allocated 12% of one processor.

Note: This calculation is always applicable, even when the LPAR runs at less than 100% capacity. If an LPAR does not use its allocation, the extra CPU cycles are reallocated based on existing weights defined to other uncapped LPARs requesting more CPU. However, capped LPARs cannot acquire more CPU cycles than their assigned weight, even if those cycles are available.

With dynamic time slicing, the LPAR weight is a guaranteed minimum, not a maximum allocation CPU resource. If all LPARs use their allotted share, this would be the amount of processing that could be performed. Normally (and in this case), very few of the LPARs have any activity. Thus, the A02 LPAR could get as much as 90% of each logical engine in its assigned time, with all of the total available free capacity, not being used by, or dedicated to, other LPARS.

8.1.3 LPAR analysis example

For example, if the weight of an LPAR is 10, and the total weights of all LPARs is 1000, then the LPAR is allocated 1% of the system. If the system consists of 10 processors, the LPAR is allocated 10% of one physical processor. If the LPAR has two logical processors, each logical processor is allocated 5% of a physical processor. Thus, increasing the number of logical processors in a complex decreases the relative speed of each logical processor in an LPAR.

8.1.4 LPAR options

LPAR shares can be defined as capped, meaning that their share of the physical system is capped to their given share. Given the situation of 1% allocated, this would be a very small amount of resources. If not capped, unused CPU resources are available to any LPAR that can utilize the resources based on given weights. Capped shares should only be used in exceptional circumstances, for example, in installations where resources need to be strictly limited, or where a financial agreement exists to provide a specific speed of processor. Otherwise, you should run uncapped, and allow PR/SM™ to allocate processor resources where they are required across the various LPARs present.

The time slice is either specific or dynamic. Specific time slices are rarely if ever used. The impact of having a specific time slice likely means erratic responsiveness from the processor subsystem. There does not seem to be any useful reason for using specific time slices.

Wait completion defines whether LPARs either give up the processor when there is no remaining work, or keep it for the remaining time slice. With wait completion enabled, an

LPAR voluntarily relinquishes the processor when its work is complete. This option may be useful for LPARs running as dedicated partitions.

8.1.5 Entire system, all partition capacity analysis

Where there are many LPARs on a system, it is important to know how fully loaded the overall system is when all of the partitions are active. A very effective way to do this is to draw a chart that uses cumulative stacking of CPU utilizations over time for all partitions. The data can be extracted from z/VM and loaded in to a spreadsheet or graphics drawing package. Note that from a VM system running in LPAR mode, you can see the CPU utilization of the other VM, VM/VSE, and z/OS LPARS, as well as others running VM with Linux guests. See the LPAR display from the VM Performance Toolkit earlier in this chapter for examples. Figure 8-2 shows that the CPU utilization absolute peaks are just over 83%, and that there is sufficient CPU headroom for workload growth on this system. It is very clear from this type of chart where the system is reaching an overall critical CPU load state, and which LPARs are mainly contributing to it.

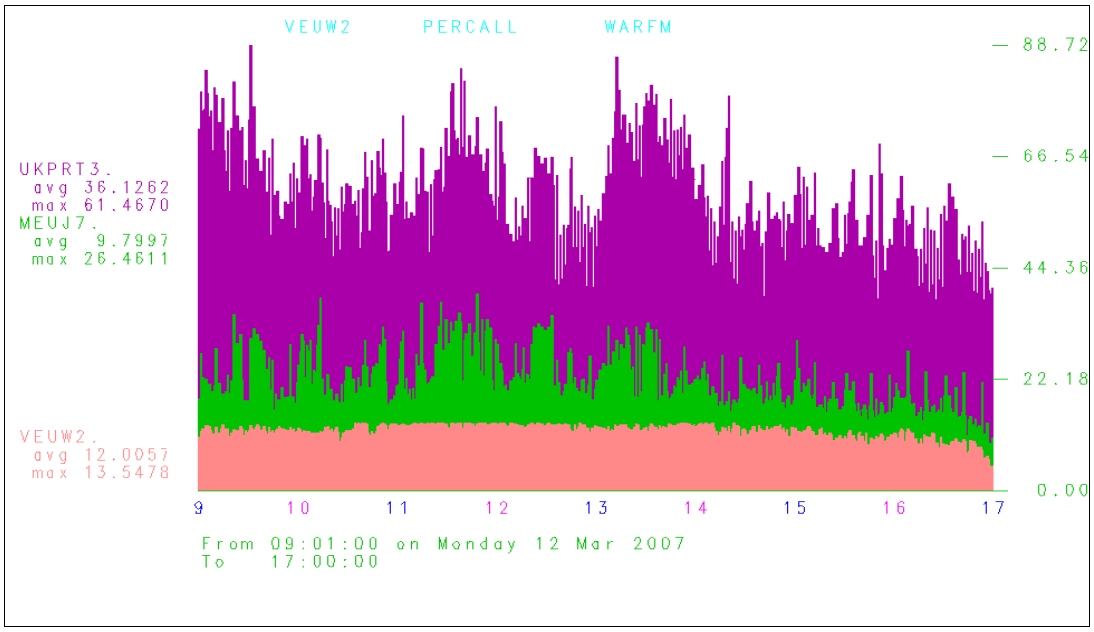


Figure 8-2 Example of cumulative stacked chart for overall system LPAR utilization

8.1.6 Shared versus dedicated processors

When there are multiple physical processors on a system to be utilized by many different logical partitions, there is the option of dedicating processors to an LPAR. In general, this is only used for two reasons:

- For benchmarks, to reduce questionable impacts from other workloads. When running benchmarks, the aim should always be to factor out interference from other work that is running in the same time frame as completely as possible. Dedicating processors means that the CPU availability is constant at all times.
- When the workload is steady enough to utilize the processors, and there are sufficient resources to justify dedicated processors. A good example is a mission-critical partition that runs at consistently high CPU utilization.

Important: When running benchmarks, for consistency reasons, always use dedicated processors to reduce the impact of other workloads on the results.

To justify the cost of System z implementations, the objective should always be high utilization. For System z, utilizations of over 90%, and approaching 100% are quite common. They are readily sustainable in workload throughput and response time terms. High utilization leverages the value of the reliability, availability, and serviceability of the System z systems. Other platforms rarely operate at high utilizations, many midrange systems peaking at 30%, and smaller server platforms at even lower levels. Dedicating physical resources such as processors to one LPAR has the potential for reducing the overall system utilization. Reducing system utilization reduces System z effectiveness and increases the cost per unit of work.

8.2 The CP scheduler

The CP scheduler function, by using time-sharing principles, attempts to keep as many of the logged-on virtual machines as possible operating concurrently. It contains many algorithms that take into account such factors as the availability of processing time, paging resources, and real storage (as compared to virtual machine requirements).

The CP scheduler uses two time slices in determining how long a given virtual machine competes for access to the processor:

- Elapsed time slice

  Virtual machines compete for use of the processor for the duration of the elapsed time slice.

- Dispatch time slice

  During its elapsed time slice, a virtual machine can only control the processor for a duration of its dispatch time slice. This is often referred to as the *minor time slice*.

When the dispatch time slice for a virtual machine expires, the scheduler readjusts its priority relative to other virtual machines competing for the processor. When its elapsed time slice expires, the scheduler drops the virtual machine from the set competing for the processor. The scheduler then attempts to add another virtual machines eligible for processor resources into the competing set.

8.2.1 Transaction classification

For dispatching and scheduling, virtual machines are classified according to their transaction characteristics and resource requirements:

- Class 1

  These virtual machines are designated as interactive tasks.

- Class 2

  These virtual machines are designated as non-interactive tasks.

- Class 3

  These virtual machines are designated as resource-intensive tasks.

- Class 0

  A special class designation for virtual machines that require immediate access to processor resources. This class, which needs a special privilege called quick dispatch, is referred to as Class 0.

For processor scheduling, started virtual machines reside on one of three lists:

- Dormant list
- Eligible list
- Dispatch list

8.2.2 The dormant list

The dormant list contains virtual machines with no immediate tasks that require processor servicing. As virtual machines move from that state and begin to require processor resources, they move to the eligible list.

8.2.3 The eligible list

The eligible list consists of virtual machines not currently being considered for dispatching due a system resource constraint (for example, paging or storage). Virtual machines are kept back in the eligible list when demand for system resource exceeds what is currently available. Virtual machines on the eligible list are classified according to their anticipated workloads requirements:

- E1

  E1 refers to class 1 virtual machines expected to perform short transactions. Upon entering the eligible list for the first time, virtual machines are classified as E1.

- E2

  E2 refers to class 2 virtual machines expected to perform medium-length transactions. These virtual machines drop to the eligible list after spending at least one elapsed time slice in E1 without completing processing.

- E3

  E3 refers to class 3 virtual machines expected to perform long-running transactions. E3 virtual machines spent at least two elapsed time slices on the eligible list without completing processing (at least one in E1 and one in E2).

Class 0 virtual machines never wait in the eligible list for processor resources. Instead, they move immediately to the dispatch list. These are classified as E0 virtual machines. We discuss E0 virtual machines in 8.4.2, “The CP QUICKDSP option” on page 117.

As processor resources become available, virtual machines are moved from the eligible list to the dispatch list. Classification on the eligible list influences the priority and elapsed time slice assigned to virtual machines by the scheduler when they move to the dispatch list. Priorities assigned in this way are intended to:

- Slow down virtual machines that are resource intensive in favor of virtual machines that require fewer resources.
- Ensure that virtual machines receive a designated portion of the processor (see 8.4.3, “The CP SET SHARE command” on page 117).
- Control the amount and type of service based on virtual machine classification (E1, E2, or E3).

We discuss how the scheduler calculates priorities in 8.3.3, “Entering the dispatch list” on page 112.

8.2.4 The dispatch list

Virtual machines contending for processor time are placed on the dispatch list. Entries higher on this list are more likely to receive processor time. Virtual machines in the dispatch list retain the transaction classification assigned while waiting in the eligible list. Transaction classifications on the dispatch list are referred to as Q1, Q2, Q3, and Q0 (directly analogous to the E1, E2, E3, and E0 classifications while on the eligible list).

Note: E0 virtual machines on the eligible list are included in the count of Q0 virtual machines displayed by the CP INDICATE LOAD command.

8.3 Virtual machine scheduling

Figure 8-3 illustrates the state transitions involved in scheduling a virtual machine.

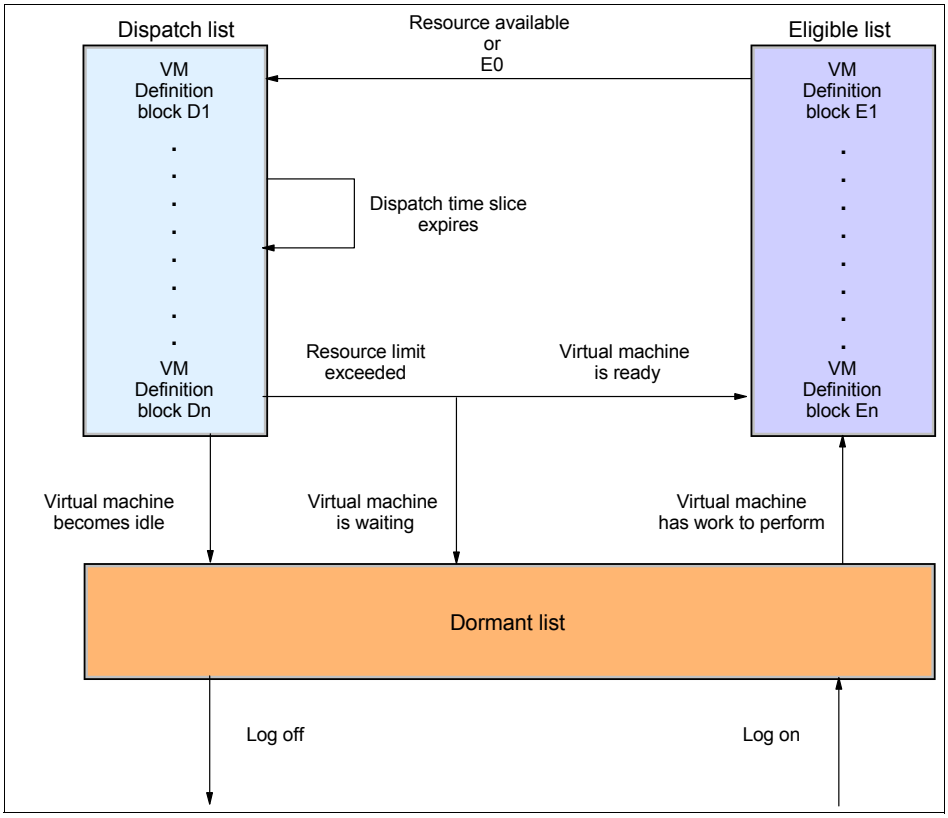


Figure 8-3 CP scheduling: state transitions

Definition: Figure 8-3 shows VM definition blocks on the dispatch and eligible lists. VM definition blocks represent virtual processors allocated to a virtual machine. By default, every virtual machine has at least one definition block (the base). Definition blocks are discussed in 8.3.4, “Scheduling virtual processors” on page 113.

8.3.1 Entering the dormant list

On logon, virtual machines are initially placed on the dormant list. Upon completing a transaction, virtual machines enter the dormant list from the dispatch list.

Note: This is referred to as *being dropped from the queue*.

A virtual machine can enter the dormant list from the dispatch list if its elapsed time slice has expired and it is waiting for a resource (such as a demand page-in). When processor time is required, virtual machines are moved to the eligible list.

8.3.2 Entering the eligible list

Upon entry to the eligible list from the dormant list, virtual machines are normally classified as E1 (E0 virtual machines move directly to the dispatch list). Virtual machines can enter the eligible list from the dispatch list if they did not complete processing in their elapsed time slice. In this case, an E1 virtual machine drops to E2, the E2 virtual machines drop to the E3, and E3 virtual machines remain in E3.

8.3.3 Entering the dispatch list

In order to move from the eligible list to the dispatch list, a virtual machine must pass three tests:

- Storage test

  The virtual machine associated working set must fit in real storage as it stands at that time. We discuss this in “The STORBUF control” on page 115.

- Paging test

  If the user is a loading user, there must be room in the loading user buffer at that time. Loading users are discussed in “The LDUBUF control” on page 114.

- Processor test

  There must be room in the dispatch buffer. The dispatch buffer size is discussed in “The DSPBUF control” on page 114.

A virtual machine on the dispatch list remains there for the duration of its elapsed time slice. Elapsed time slices are assigned to virtual machines as they enter the eligible list and are based on their transaction class.

CP assigns a initial elapsed time slice value of 1.2 seconds for E1 virtual machines during initialization. This value is dynamically adjusted during system operation:

- As the number of virtual machines that complete processing in E1 increases, the size of the E1 elapsed time slice decreases.
- As the number of E2 virtual machines increases, the size of the E1 elapsed time slice increases.
- If the number of E1 virtual machines drops below a threshold, and the number of E3 virtual machines increases above a threshold, the size of the E1 elapsed time slice increases.

The E1 elapsed time slice will always be a value between 50 ms and 16 seconds. E0, E2, and E3 elapsed time slices are assigned values 6, 8, and 48 times larger, respectively, than the E1 time slice.

The dispatch time slice is computed at CP initialization. Its value can be queried using the QUERY SRM DSPSLICE command.

While on the dispatch list, a virtual machine runs for its designated elapsed time slice. If processing has not completed in that period, it is moved to back to the eligible queue (with a lower workload classification if not already in the E3 queue).

8.3.4 Scheduling virtual processors

Additional VM definition blocks are created for each virtual processor defined to a virtual machine. These additional blocks are linked to their respective base definition block. Both the base and any additional definition blocks cycle through the three scheduler lists. The base definition block owns the virtual storage and most virtual resources for the virtual machine. As the base and additional definition blocks move through the lists, the base is always in a list at least as high as any of its linked definition blocks.

Note: The hierarchy of lists is defined as (highest to lowest):

1. Dispatch list
2. Eligible list
3. Dormant list

This ensures that the resource requirements of the virtual machine as a whole are considered when scheduling a VM definition block. However, processor time consumed by any additional definition blocks is measured independently of the base. This value is used when scheduling that block while on the dispatch list.

To illustrate, consider a virtual machine with two virtual processors (and two VM corresponding definition blocks), both of which start in the dormant list. If the additional definition block becomes dispatchable, both definition blocks (the base and the additional) move to the eligible list. After waiting in the eligible list, both definition blocks again move to the dispatch list and are given the same priority. As the additional definition block consumes resources, however, its intermediate dispatch priority is calculated independently from the base block definition. We discuss virtual processors in more detail in 9.6, “Performance effect of virtual processors” on page 140.

8.3.5 z/VM scheduling and the Linux timer patch

The Linux kernel keeps track of time using a timer that interrupts at a constant rate (System z at 100 Hz). The Linux kernel maintains a global variable called jiffies, which is updated. A variable jiffies is updated and queues are inspected for work to be done. In a Linux guest running on z/VM this behavior interferes with the z/VM scheduler model. With the 10 ms timer interrupt, CP classified a Linux virtual machine as a long-running task and assigns it to Q3. It prevents z/VM from distinguishing between running and idling guests.

Current distributions allow you to switch the timer interrupts on and off on demand. With the on demand timer patch enabled, Linux virtual machines are dispatched less frequently when they are idling. This significantly reduces the number of Q3 servers competing for resources. Reducing the concurrent number of tasks competing for resources reduces the contention felt by the shorter tasks. For more details about the solution see 9.3, “The Linux timer modification” on page 133.

8.4 CP scheduler controls

The CP scheduler can be influenced using two general types of controls:

- SRM controls globally influence the overall processor resources.
- Local controls influence how an individual virtual machine is regarded by the scheduler.

8.4.1 Global System Resource Manager (SRM) controls

Overall z/VM processor resources can be tuned using the CP SET SRM command.

The DSPBUF control

The CP SET SRM DSPBUF command controls the number of users in the dispatch list. It permits you to overcommit or under-commit processor and I/O device resources. The command format is:

```
SET SRM DSPBUF i j k
```

Where:

i	Specifies the number of dispatch list slots available for E1, E2, and E3 users
j	Specifies the number of dispatch list slots available for E2 and E3 users
k	Specifies the number of dispatch list slots available for E3 users

Note: Valid operand ranges are $32767 \geq i \geq j \geq k \geq 1$.

For example:

```
SET SRM DSPBUF 35 30 18
```

This command allocates 35 slots on the dispatch list:

- Five are guaranteed to E1 users (35–30).
- Thirty are available to E2 and E3 users, 12 of which (30–18) are guaranteed not to be occupied by E3 users.
- Eighteen are available to E3 users (although these may be occupied by E1 and E2 users).

Important: Using DSPBUF to control dispatch lists is a very risky process. We do not recommend making adjustments to this control. It is better to allow CP algorithms to use dynamic control, dependent on the changing system conditions.

The LDUBUF control

The CP SET SRM LDUBUF command partitions the commitment of the system's paging resources.

Definition: LDUBUF stands for loading user buffer. A loading user is a heavy user of paging resources and is expected to have a high paging rate. A loading user is defined as a user that takes five page faults within one time slice of work. Because a time slice is relatively very small, any user that takes five page faults within that time is totally consuming the equivalent of one paging device. Use the INDICATE QUEUES EXP command to display the current loading users.

The command format is:

```
SET SRM LDUBUF i j k
```

Where:

i	Specifies the percentage of system paging resources available to E1, E2, and E3 users
j	Specifies the percentage of system paging resources available to E2 and E3 users
k	Specifies the percentage of system paging resources available to E3 users

The STORBUF control

The CP SET SRM STORBUF command partitions the commitment of real storage in terms of pages based on the transaction class (E1, E2, E3) of a user. This command enables you to overcommit or under-commit real storage.

The command format is:

```
SET SRM STORBUF i j k
```

Where:

i	Specifies the maximum percentage of system storage available to E1, E2, and E3 users
j	Specifies the maximum percentage of system storage available to E2 and E3 users
k	Specifies the maximum percentage of system storage available to E3 users

Note: Valid operand ranges are $999 \geq i \geq j \geq k \geq 0$. The default values for STORBUF are 125%, 105%, and 95%, respectively.

Performance gains might be realized by overcommitting the overall system real storage when expanded storage is available. When you overcommit storage this way, a virtual machine's working set is still computed as before, but the apparent real storage available to virtual machines who want to enter the dispatch list appears larger. Therefore, more virtual machines are allowed into the dispatch list. This might result in higher paging rates, but often the benefit of reducing the eligible list and moving users into the dispatch list offsets this increase. Expanded storage acts as a very effective, fast page buffer to real storage.

The CP scheduler perceives the storage requirement for virtual machines that do not drop from queue to be much larger than actual. Because the scheduler cannot determine the real storage requirements, in this case, raising the STORBUF control can improve system performance. This effectively disables the storage test discussed in 8.3.3, "Entering the dispatch list" on page 112.

For Linux guests, setting the STORBUF control has been an inexact science. There was little difference in a STORBUF value of 900, as opposed to a value of 300. In both cases, the storage test was likely disabled. Conversely, when controlling a large population of CMS users, overcommitting real storage using STORBUF is a very effective and desirable control mechanism. With improvements to the way Linux runs in guest mode, particularly the timer modification, it is worth experimenting with STORBUF overcommittment.

The MAXWSS control

The CP SET SRM MAXWSS command sets the maximum working set that a normal user on the dispatch list is allowed to have. If the user's working set size exceeds this percentage, that user is dropped back into the eligible list from the dispatch list. The command format is:

```
SET SRM MAXWSS m
```

Where *m* specifies the maximum percentage of system pageable storage available to a user.

MAXWSS is intended to prevent large virtual machines from acquiring a greatly excessive amount of system memory at the expense of smaller users.

Note: In order for this parameter to work as intended, virtual machines must reside in the eligible list (that is to say, the scheduler must consider the system to be storage constrained, based on the STORBUF settings).

The DSPSLICE control

A virtual machine is assigned a dispatch time slice each time it is dispatched. The dispatch time slice is calculated at system initialization based on real processor speed and represents a fixed number of instructions processed. This value can be changed using the CP SET SRM DSPSLICE command:

```
SET SRM DSPSLICE min
```

Where *min* specifies the minimum dispatching time slice (in milliseconds).

Note: Valid dispatching time slice values are in the range $100 \geq min \geq 1$.

When a virtual machine is first dispatched, it runs until:

- Its minor time slice interval expires.
- It completes the workload and proceeds to wait for more work.
- It enters CP mode.
- An interrupt occurs.

When redispatched, the virtual machine is assigned a new dispatch time slice. (If it relinquished the processor due to an interrupt, it is assigned the remaining portion of the previous dispatch time slice.)

When to use SRM controls

SRM controls can be used to ensure that:

- Processor resources are utilized as much as possible.
- Thrashing situations do not occur.
- When performance is really bad (and this *will* sometimes happen), some servers get preferential access to processor resources.

For example, if performance is very bad, you would still want TCP/IP (or perhaps a DNS server or security manager) to perform well. Thus, it is preferable to utilize the SRM controls properly and not overuse the QUICKDSP option.

Important: The QUICKDSP option should be used for machines that need to run when things are very bad. Set QUICKDSP to the select few that must absolutely perform well, such as virtual machines that run mission-critical processes.

You should use STORBUF to over-allocate storage when virtual machines do not drop from the queue. In this situation, the dispatching and scheduling algorithms become skewed. The true storage requirement is unknown, and proper scheduling is impossible. Installations have found that when running many servers (even *idle servers* that never drop from queue), the perceived storage requirement is very high. Using high STORBUF values to *over-allocate* storage can improve this situation.

In an installation where the virtual machines do drop from queue, using STORBUF to over-allocate storage might not be appropriate.

Note: In order to drop from queue, Linux guests must run with the timer modification installed (see 8.3.5, “z/VM scheduling and the Linux timer patch” on page 113) *and* use an appropriate network architecture (see 11.2.1, “Qeth device driver for OSA-Express (QDIO)” on page 173).

For this case, if there is expanded storage, the SET SRM XSTORE command should be set to at least 50%, or even to its maximum value.

In environments where there is considerable paging, paging should be controlled using the proper SET SRM LDUBUF command. The default LDUBUF in a paging environment allows the paging subsystem to be 100% consumed (see 8.5, “Analysis of the SET SRM LDUBUF control” on page 118). At the point where the paging subsystem is at 100% utilization, by definition, there would not be much value by increasing the LDUBUF and allowing more users to consume paging resources.

In 8.5, “Analysis of the SET SRM LDUBUF control” on page 118, we examine the operation of some SRM controls.

8.4.2 The CP QUICKDSP option

The SET QUICKDSP command does one thing only, but it does that very well. It should be used for service machines that are required to run, even when there are serious memory or processor constraints.

The CP scheduler classifies virtual machines assigned the QUICKDSP option as E0 virtual machines. An E0 virtual machine is added immediately to the dispatch list whenever it has work to do, without waiting in the eligible list. The CP scheduler bypasses the normal storage, paging, and processor tests when moving E0 virtual machines from the eligible list to the dispatch list.

8.4.3 The CP SET SHARE command

VM uses the SHARE command to establish resource levels for the various CMS users, guests such as Linux, and service virtual machines. The first decision when using SHARE is whether to use ABSOLUTE or RELATIVE. Then you must decide on the amount of that SHARE type that you wish to allocate to each given user. This involves consideration of the entire group of users residing on the system, at the outset after system creation, and each time a change to a SHARE is made.

The simplest way to decide which to use for a specific server is to answer the question: As more users log on to this system, should this service machine get more CPU or less?

- A relative share says that this server should get a relative share of the processor, relative to all virtual machines in the dispatch and eligible lists. As more users log on, the share drops. Although the exact priority for a given user is indeterminate, this works very well

where, for example, a large community of CMS users need a fair share of the finite resources. A relative share evolves with the changing numbers of users, making way for newly logged on users, and also ensuring that the fair sharing process is maintained when users stop using the system.

- Absolute shares remain fixed up to the point where the sum of the absolute shares is 100% or more, then all of the absolute shares for each user are established by using the sum of all absolute shares allocated on the system as the divisor. An absolute share of 10% equals 10/100%, until more than 100% is allocated across the system. If 120% is allocated, the 10% share is then equal to 10/120%.

Servers such as TCP/IP or even DNS servers might have a requirement that rises as the level of work increases. These servers should have absolute shares. Some other service virtual machines also have load-related requirements, for example, the CP monitor produces more records as more users log on. You should consider giving small absolute shares to MONWRITE and the Performance Tool Kit. Generally, all other users should use relative. More important service machines not needing an absolute share, or more important users, should be given a higher relative share than the default value of 100.

VM allocates resources at run time by considering the absolute share holders first, then moving to the relative share holders. Also note that all the shares relate to the currently scarce resource. Though this is generally CPU, there are occasions when other key resources are constrained.

The size of the share is both a business decision and a performance decision. For example, if one server is assigned a very high share, which might be as much of the system as the rest combined, one expects this server to be absolutely critical to your business. This server has the capability of taking resources whenever it needs them, but if this server starts looping, it could easily consume all the resource allocated.

A simple way of looking at share values is, if there is heavy contention for the processor, what servers would you like to run to ensure that you are meeting your mission goals?

TCP/IP, with such a generally key role in the network infrastructure, is an obvious candidate, as are required services such as Domain Name System servers. TCP/IP and other such required servers should have absolute values that approach the CPU consumption that would be required at peak loads. There is no value in giving more than that. Use your peak loading performance data and aim to review the overall system settings periodically.

8.5 Analysis of the SET SRM LDUBUF control

The purpose of the SET SRM LDUBUF command is to control page thrashing. Thrashing is a situation where paging is at a point where less work is being accomplished because of contention for paging and storage devices. The following analysis shows some of the key points in evaluating the use of LDUBUF.

When the loading capacity is evaluated by the scheduler, the number of paging devices is the *loading capacity*. Setting LDUBUF to the default of 100 85 65 allows 100% of the total capacity (the number of paging devices) to be utilized. Thus, if there were seven paging devices, seven users that were considered *loading* would be allowed onto the dispatch list. Additional users would be delayed on the eligible list until one or more of the existing loading users either dropped from the list or obtained their working set in storage and stopped causing page faults.

For this analysis, a benchmark was used that would allocate storage, perform a function, and release the storage, which is typical of many Linux environments. What the measurements

show is that LDUBUF at the default level allows the total paging subsystem to be 100% consumed.

8.5.1 Default setting analysis

Figure 8-4 shows CPU utilization and paging rates using the default settings for LDUBUF and STORBUF.

```
FCX225      CPU 2094  SER 2991E  Interval 08:53:56 - 14:22:56    Perf. Monitor

          <------- CPU --------> <Vec> <--Users--> <---I/O---> <Stg> <-Paging--
               <--Ratio-->                          SSCH  DASD Users <-Rate/s--
Interval    Pct        Cap-  On-   Pct  Log-       +RSCH  Resp    in PGIN+ Read
End Time   Busy   T/V  ture line  Busy   ged Activ    /s  msec Elist PGOUT Writ
>>Mean>>    5.9  1.07 .9672  4.0  ....    20    10  19.9   2.7    .7  79.7  12
12:08:56   22.3  1.01 .9925  4.0  ....    20    10  38.9   2.9    .0    .5   1
12:09:56   46.9  1.01 .9950  4.0  ....    20    12  40.2   5.0    .0    .5
12:10:56   28.9  1.01 .9939  4.0  ....    20    11  28.9   5.3    .0   1.0   1
12:11:56   26.2  1.02 .9893  4.0  ....    20    12  18.9   5.6    .0  25.4   6
12:12:56   30.2  1.01 .9924  4.0  ....    20    10 130.6   5.5    .0 112.6  38
12:13:56   43.5  1.01 .9956  4.0  ....    20    11  61.5   4.2    .0    .3   4
12:14:56   40.5  1.01 .9946  4.0  ....    20    10  25.0   2.0    .0    .3   1
12:15:56   38.3  1.02 .9931  4.0  ....    20    11  39.5   1.7    .0    .5   1
12:16:56   36.9  1.03 .9916  4.0  ....    20    11  45.7   1.3    .0    .5   2
12:17:56   38.7  1.03 .9909  4.0  ....    20    11  53.4   1.2    .0    .6
12:18:56   28.8  1.02 .9932  4.0  ....    20    10  21.8   1.7   1.0    .3   2
12:19:56   25.2  1.01 .9946  4.0  ....    20    10  12.4   2.4   1.0   2.3   3
12:20:56   25.0  1.01 .9942  4.0  ....    20     9  24.7   2.4   1.0    .4
12:21:56   25.0  1.01 .9944  4.0  ....    20    11  22.5   2.2   1.0    .9   3
```

Figure 8-4 CPU and paging with default LDUBUF/STORBUF settings

The report shows CPU utilization varied from 22.3% to 46.9% with a paging rate to DASD of 112.6 pages/second. The paging to expanded storage was 400–500 pages/second.

8.5.2 User queue analysis

Figure 8-5 shows a queue analysis for a benchmark class of users named IUCVRO. These users normally drop from the queue when idle, as they use the IUCV driver and have the timer patch applied (as discussed in 11.2.1, “Qeth device driver for OSA-Express (QDIO)” on page 173).

```
FCX101      CPU 2094  SER 2991E  Interval 08:54:00 - 14:48:56    Perf. Monitor

TIME >OGN  ACT TR-T NT-T C1ES TR-Q NT-Q TR/S NT/S  ITR %PQ %IQ %LD %EL  Q1  Qx
14:31  21   12  .53  .87 2.48   .7   .3  1.4   .4 55.2   0   0   0   0   2   4
14:32  21   10  .80  .36 2.42  1.0   .0  1.3   .2 49.0   0   0   0   0   2   4
14:33  21   11  .82  .59 2.38  1.0   .1  1.3   .3 57.0   0   0   0   0   1   5
14:34  21   10  .68  .48 2.33   .9   .1  1.4   .3 59.5   0   1   0   0   2   5
14:35  21   12  .66  .25 2.27   .9   .0  1.4   .3 55.6   0   1   0   0   1   4
14:36  21   10  .56  .33 2.20   .8   .0  1.6   .2 64.3   0   0   0   0   1   4
14:37  21   11  .47  .63 2.15   .7   .1  1.6   .3 51.4   0   0   0   0   2   4
14:38  21   10  .52  .50 2.09   .7   .1  1.5   .3 45.5   0   1   0   0   1   3
14:39  21   11  .49  .59 2.03   .7   .1  1.5   .3 59.9   0   0   0   0   2   4
14:40  21   12  .53  .65 1.99   .7   .2  1.5   .3 55.0   0   1   0   0   1   4
14:41  21   11  .48  .58 1.95   .7   .2  1.5   .4 53.6   0   1   0   0   0   5
14:42  21   10  .66  .90 1.92   .8   .3  1.3   .4 51.2   0   0   0   0   1   6
14:43  21   11  .45 1.00 1.89   .6   .4  1.3   .4 59.9   0   1   0   0   3   4
14:44  21   10  .48  .66 1.85   .6   .2  1.4   .4 56.7   0   0   0   0   1   3
14:45  21   12  .45  .17 1.79   .7   .0  1.7   .3 55.5   0   0   0   0   2   4
14:46  21   10  .53  .19 1.74   .8   .0  1.6   .3 67.8   0   1   0   0   3   5
14:47  21   11  .66  .11 1.69  1.0   .0  1.5   .3 60.4   0   0   0   0   2   5
14:48  21   10  .59  .03 1.64   .9   .0  1.6   .3 61.5   0   0   0   0   2   5
```

Figure 8-5 User queue analysis

At this time, of the 21 logged on, only between 10 and 12 users are active, and between 4 and 8 users are in queues (Q1+Qx).

8.5.3 DASD analysis

Figure 8-6 shows an analysis of a Direct Access Storage Device (DASD) during the default run.

```
FCX109      CPU 2094  SER 2991E  Interval 17:51:56 - 17:52:56    Perf. Monitor

Page / SPOOL Allocation Summary
PAGE slots available      1940916          SPOOL slots available       600840
PAGE slot utilization           3%         SPOOL slot utilization         14%
T-Disk cylinders avail.         0          DUMP slots available            0
T-Disk space utilization      ...%         DUMP slot utilization          ..%

____ .            .                          .     .     .     .     .      .
< Device Descr. ->                        <------------- Rate/s ------------->
            Volume Area   Area      Used  <--Page---> <--Spool-->         SSCH
Addr Devtyp Serial Type   Extent       %  P-Rds P-Wrt S-Rds S-Wrt Total  +RSCH
CD31 3390   LX6RES DIRECT    1-  20   30    ...   ...   ...   ...   ...    ...
CD34 3390   LX6SPL SPOOL     1-3338   14     .0    .0    .0    .0    .0     .0
CD36 3390   LX6PG1 PAGE      1-3338    3     .0   3.5   ...   ...   3.5     .3
CD38 3390   LX6PG3 PAGE      0-3338    2     .0    .0   ...   ...    .0     .0
CD41 3390   LX6PG4 PAGE      0-3338    2     .0    .0   ...   ...    .0     .0
CD49 3390   LX6PG2 PAGE      1- 767    9     .0   2.6   ...   ...   2.6     .3
```

Figure 8-6 Analysis of paging devices

The four page devices on this system appear in the CP Owned Disks table in the VM Performance Toolkit. As this system has 4 GB of main storage and 1 GB of expanded

storage, in the example, the page devices have slot utilizations of between 3% and 9%, with a current page rate of just 6.1 pages/sec.

Figure 8-7 shows an analysis of the channel subsystem.

```
FCX131      CPU 2094  SER 2991E        Status   15:05:56         Perf. Monitor

<----  Ranges  ---->   Device      <- Channel Path Ids  ->   Control
Device-No  Subch.-ID   Type         1  2  3  4  5  6  7  8   Unit      Status
2040-204E  06DE-06EC   OSA          00 .  .  .  .  .  .  .   1731-01   Online
204F       06ED        OSA          00 .  .  .  .  .  .  .   1731-01   Online
20A0-20AE  070E-071C   OSA          03 .  .  .  .  .  .  .   1731-01   Online
20AF       071D        OSA          03 .  .  .  .  .  .  .   1731-01   Online
22A0-22AE  3795-37A3   OSA          0D .  .  .  .  .  .  .   1731-01   Online
22AF       37A4        OSA          0D .  .  .  .  .  .  .   1731-01   Online
5A90       086E        CTCA         95 .  .  .  .  .  .  .   3088      Online
5A98-5A99  0872-0873   CTCA         97 .  .  .  .  .  .  .   3088      Online
C700-C709  2526-252F   3390-9 (E)   8C 8D 8E 8F .  .  .  .   2105-E8   Online
CD31-CD49  2B57-2B6F   3390-3 (E)   80 9D 82 A5 .  .  .  .   2105-E8   Online
E400-E40F  349E-34AD   OSA          F0 .  .  .  .  .  .  .   1731-05   Online
F003       3559        3270-3       12 .  .  .  .  .  .  .   3174-1D   Online
F036-F037  358C-358D   3270-2       12 .  .  .  .  .  .  .   3174-1D   Online
F083       35D1        3270-2       13 .  .  .  .  .  .  .   3174-1D   Online
F0B6-F0B7  3604-3605   3270-2       13 .  .  .  .  .  .  .   3174-1D   Online
```

Figure 8-7 Analysis of channel subsystem

If adding page devices were an option, there should be sufficient channel capacity to support additional devices. In this DASD configuration, all of the devices, including several paging devices, are in two device number ranges that each share four channel paths (CHPIDs). The paging devices are in the range CD31-49, and use CHPIDs 80, 9D, 82, and A5.

Now that we know what channel paths are being used, we evaluate the capacity of the channels being utilized in Figure 8-8.

```
FCX161      CPU 2094  SER 2991E  Interval 08:54:56 - 17:23:56    Perf. Monitor

CHPID Chan-Group       <%Busy>  <----- Channel %Busy Distribution 08:54:56-17:
(Hex) Descr Qual Shrd  Cur Ave  0-10 11-20 21-30 31-40 41-50 51-60 61-70 71-80
 80   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 81   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 82   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 9D   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 9F   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 A0   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 A1   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 A2   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 A4   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
 A5   FICON   00 Yes     0   0  100     0     0     0     0     0     0     0
```

Figure 8-8 Analysis of channel capacity

This chart shows that CHPIDs 80, 9D, 82, and A5 all have channel busy utilizations in the 0–10% range. There is plenty of available channel bandwidth for the paging devices.

Main storage can typically range from a few to tens of GB, but it is always very important that the paging subsystem is able to withstand the loads that will be placed on it as a system becomes more heavily loaded and begins to page at increasingly high rates.

8.6 Virtual Machine Resource Manager

Virtual Machine Resource Manager (VMRM) was introduced in z/VM 4.3. It runs in a service virtual machine (VMRMSVM) and dynamically manages the performance of *workloads*. VMRM uses a set of user-specified workload definitions and goals, compares these with the achieved performance, and makes adjustments accordingly. (It is conceptually somewhat similar to the Workload Manager used by z/OS.) The basic idea is to allow performance objectives to be set in a manner more closely aligned with business objectives than has been the case previously.

VMRM is only effective in a constrained environment because it works by prioritizing workloads in order to enhance their access to resources, and accordingly restricts other work's access to those resources. If the z/VM system is not constrained, VMRM cannot enhance access to resources because they are readily available anyway.

A *workload* is a collection of one or more virtual machines that are to be treated as an entity for the purpose of performance management. Each workload has certain goals defined for it. These goals are for DASD and CPU utilization. A workload can have a DASD goal, a CPU goal, or both. A goal represents the relative importance of a workload for the particular installation. Configuration changes can be made at any time, but require the VMRM service machine to be stopped and restarted to make them effective.

VMRM uses CP MONITOR data to determine the level of activity on the system and whether workloads are meeting the defined goals. VMRM allows some latitude, 5%, in making this determination (this helps avoid overreaction when a workload is near its goals).

Every minute (the default) VMRM examines the system and workloads and picks a workload that is failing to meet its goals. If it finds one, it then uses the CP SET SHARE and CP SET IOPRIORITY commands to adjust the workload to give it more access to the resource or resources for which the goals were not met. VMRM remembers which workloads it adjusted recently and does not alter them again for a while. Instead, it might choose another workload to adjust at the next interval.

VMRM cannot adjust users with fixed CPU or I/O priorities. If you have defined users with absolute or hard limited SHARE, or absolute I/O priority, this has to be changed for VMRM to manage them. This also allows specific users that otherwise would be treated as part of a workload to be effectively excluded from it, but it would normally be more sensible not to define such a user as part of a workload in the first place.

8.6.1 Implications of VMRM

Because VMRM sits there adjusting performance parameters at frequent intervals, it is able to adjust to changing conditions. As the workload on the z/VM system changes over the course of a day, VMRM can make adjustments to the defined (and running) workloads in order to try to make them meet the goals defined for them. (It cannot ensure that workloads meet their goals because it cannot create resources out of thin air.)

Because VMRM adjusts z/VM tuning parameters, it might conflict with manual efforts to tune z/VM. In particular, manual use of the CP SET SHARE and CP SET IOPRIORITY commands is likely to cause problems, or at least unclear results.

Support for I/O priority queuing was added to z/VM to support VMRM. This is probably as significant as the addition of VMRM itself.

CP SET IOPRIORITY and the associated directory statement were added to z/VM to support VMRM. However, these can also be used for manual tuning instead.

Because VMRM uses CP MONITOR data, it might be affected by other users of the CP MONITOR data, and vice versa. As is always the case when more than one thing uses CP MONITOR data, some care in setup is required.

As a result of its dynamic nature of making changes relative to currently varying conditions, VMRM makes benchmarking difficult. Benchmarking typically requires repeatable conditions, and the continual adjustments of VMRM make repeatable conditions unlikely.

VMRM is not directly influenced by memory usage and has no direct effect on memory usage.

8.6.2 Further information about VMRM

For further information about VMRM, refer to Chapter 17 of the *z/VM Performance manual, Order Number*, SC24-610903.

8.7 CPU time accounting

Since Linux was ported to run on an IBM mainframe hardware platform, there has been an outstanding issue with the way that it collects and accounts utilization data. Traditionally, Linux ran as the exclusive operating system on a discrete, distributed hardware server, and the only requirements that were built into it reflected that role. When running on a mainframe, it becomes a virtual server environment, sitting on other virtual layers, a fairly radical shift, and so we should expect that some fundamental changes would be necessary. When, for example, a user issued the very common **`top`** command, the utilization numbers produced could sometimes be at considerable variance from the true values.

In the new operating space, Linux had moved from the environment it had previously relied on to keep track of what was really happening in time. Linux utilizes a tick-based CPU time accounting mechanism. When a time interrupt takes place, every 1/100th of a second on a System z system, Linux discovers exactly which context has caused the interrupt, and charge accounts the total time slice to this context. For a kernel interrupt, the entire time slice is accordingly accounted as system time. In the following examples, the top bar shows how Linux would account in a freestanding machine, and the accounted figures of 2/100th for kernel and 7/100 for the user are correct. Compare and note how in the second and third bars, with the introduction of the 1/100th second interrupt, the gross timing inaccuracy of when the context switches means there will be a varying level of deviation from the true values, depending on exactly when in time the ticks commence.

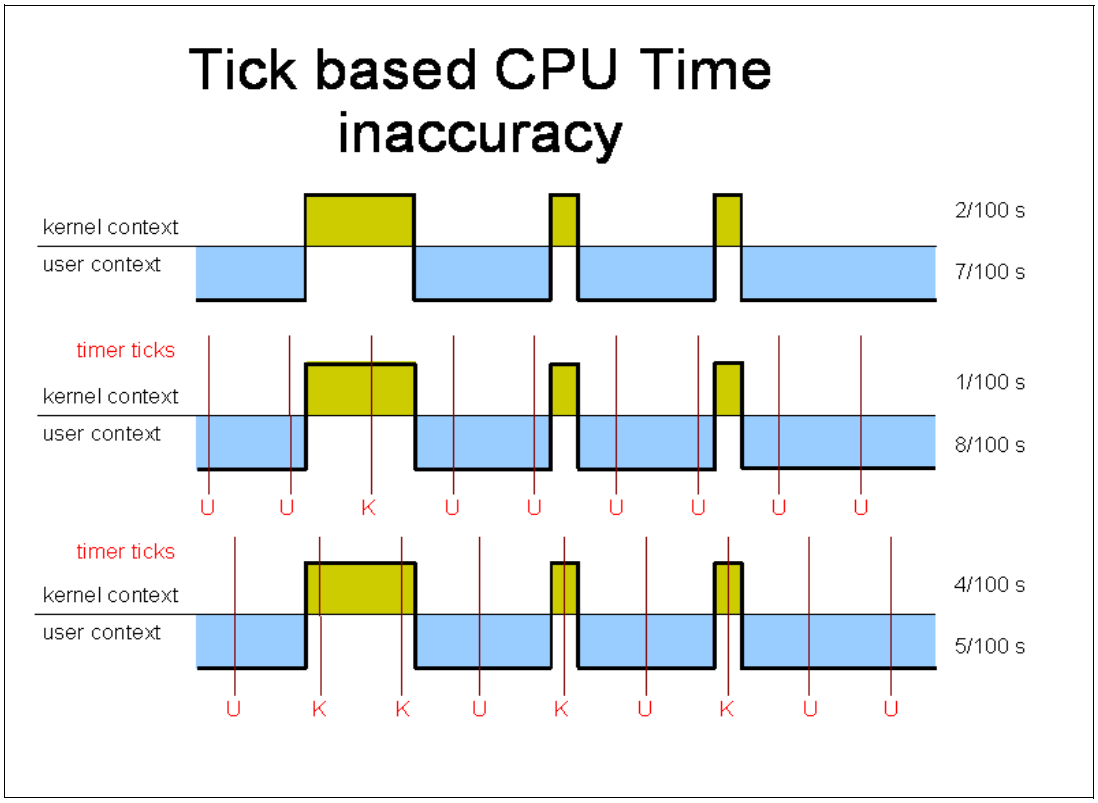


Figure 8-9 Tick-based CPU time inaccuracy

This tick-based CPU time accounting is, by its nature, imprecise. When running on free-standing, non-virtualized servers, it usually does the required job, as over time, any inaccuracies tend to be balanced out. However, when running on z/VM the situation is different. Now the Linux system is running on virtual CPUs, where more virtual than real CPUs may be in use. The underlying time slicing concept skews the normal Linux time sensing, as it believes that it has control of the virtual CPU at all times, whereas in fact, the processor control is being spread amongst all of the active virtual machines. Linux will therefore lose track of its normal time base, and as the chart shows, as context switching takes place, the accounting is highly inaccurate. This can lead to the CPU utilizations reported from Linux becoming highly inflated from their true values, when related to the actual time Linux was actually being dispatched. Also, the way the accounting is attributed may be incorrect. While Linux believed it was running on logical processor, it might have accounted time to a process that could not possibly use it. A different approach for accounting on virtual systems is required if the results are going to be accurate.

8.7.1 Virtual CPU time accounting and steal time

In order to improve the accuracy of CPU time accounting on virtual systems, the mechanism must be able to both distinguish between real and virtual CPU time, and recognize the concept of being in an involuntary wait state at some times. These wait states are referred to as *steal time*. The steal time is represented by the white space gaps within the shaded horizontal bars after the context is switched to kernel or user, respectively. This new methodology has been introduced on the System z platform. It dispenses with the old Linux, elapsed time base, and replaces it with the System z virtual CPU timer. See Figure 8-10.

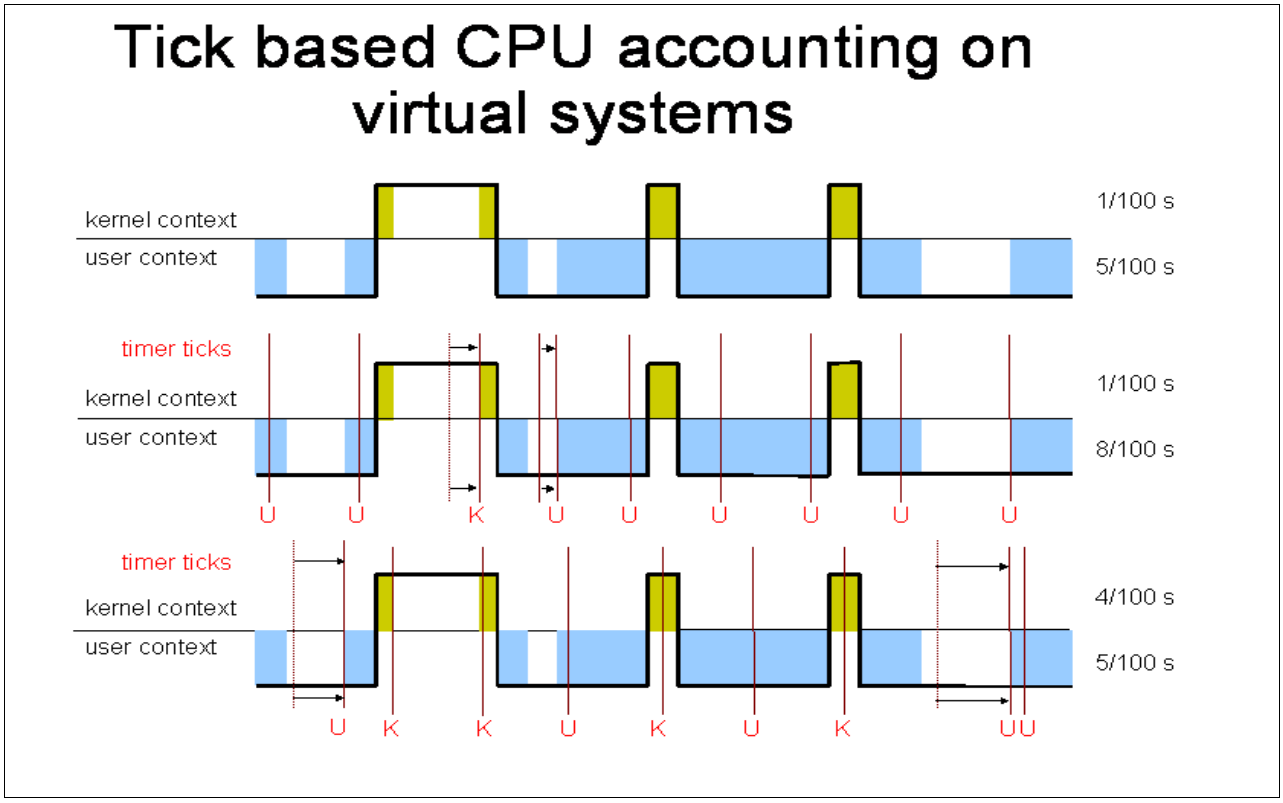


Figure 8-10 Tick-based CPU accounting

The Linux distributions Novell SUSE, SLES10, and Red Hat RHEL5 use the new virtual CPU accounting by default. However, they can also be configured to use the older method. When using the new method, Linux basis commands that display Linux process time data, such as time, ps, top, and vmstat, now show precise CPU utilization times. See Figure 8-11.

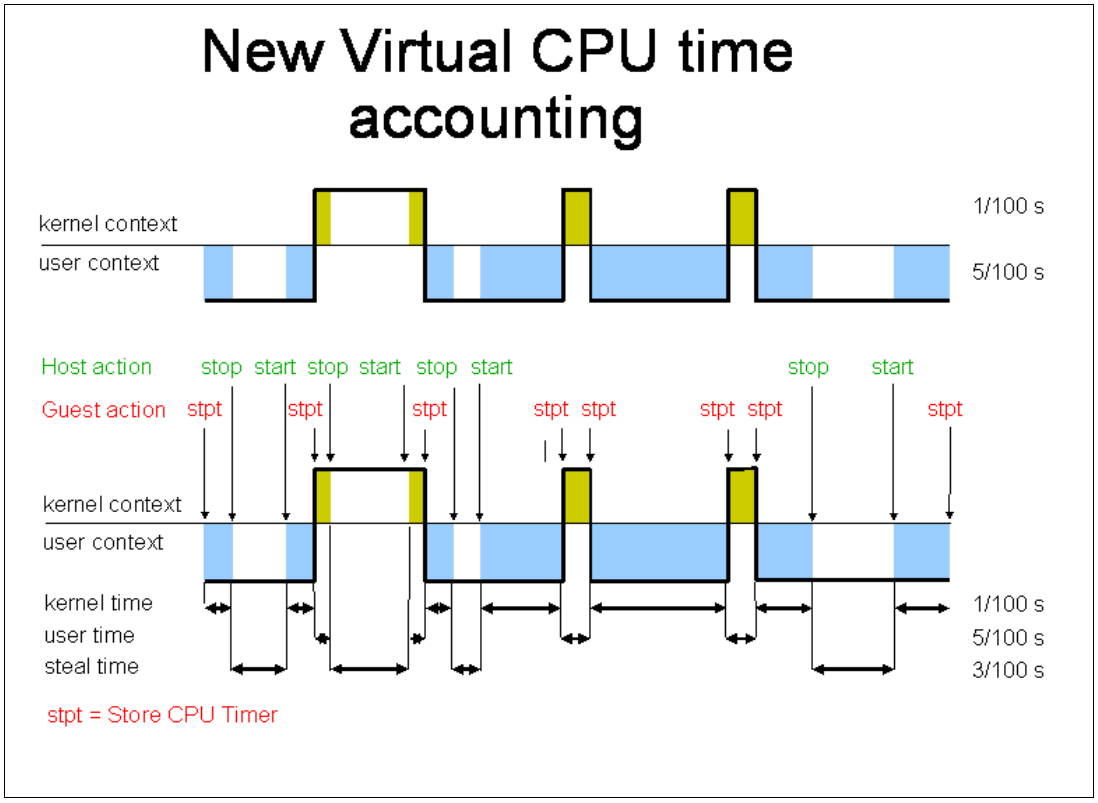


Figure 8-11 New virtual CPU time accounting

This shows a Linux on System z where the new virtual CPU time accounting mechanism has been implemented. It precisely aligns the actions taking place within Linux with the System z TOD (time of day) clock. Each CPU has its own timer, and the stepping rate is synchronized with the TOD clock, but it is only incremented when a virtual CPU is backed by a physical CPU, so Linux will have precise knowledge of when it is being dispatched on a time slice, or not. The stpt instruction (store CPU timer) is added to the Linux system call path. By storing the CPU timer, the CPU time actually used can now be accurately computed for a Linux guest, as shown in Figure 8-12.

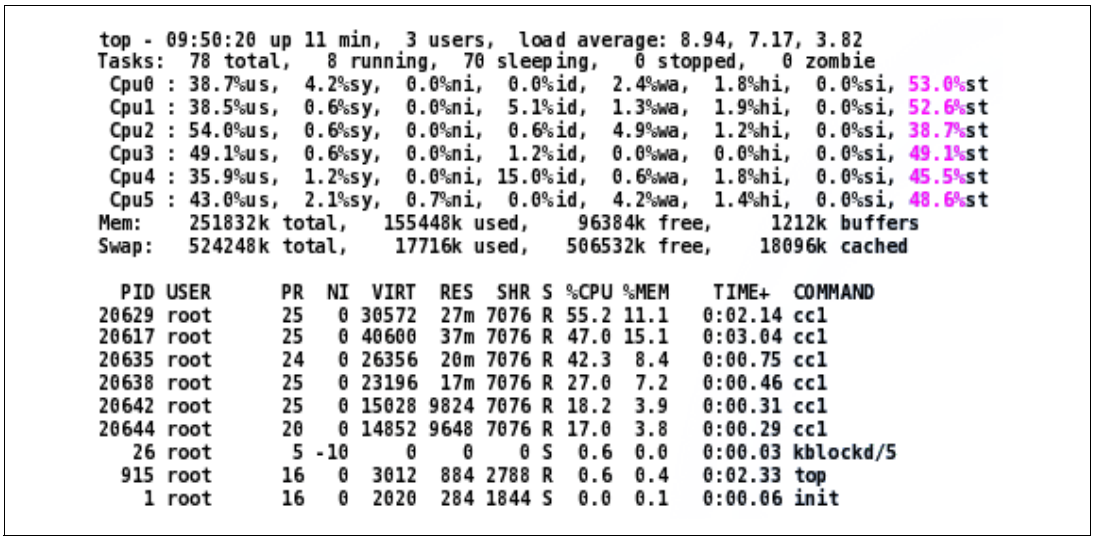


Figure 8-12 Sample of new TOP output

Note the steal time is shown in the right-most column in this example.

9

Tuning processor performance for z/VM Linux guests

This chapter discusses tuning processor performance for Linux guests. Topics include:

- Processor tuning recommendations
- The effect of idle servers on performance
- The Linux timer modification
- OSA-Express QDIO enhancements for Linux under z/VM
- Infrastructure cost
- Performance effect of virtual processors

© Copyright IBM Corp. 2003, 2008. All rights reserved.

9.1 Processor tuning recommendations

CPU time is limited to the number of available processors. Unnecessary loss of CPU resources is very common, and so steps should be taken to reduce processor requirements:

- Eliminate unnecessary Linux services.

  Default Linux guest installations typically start services that probably are not used. These services consume CPU cycles even if no useful work is performed. Remove unneeded services from the Linux startup sequence.

- Remove unneeded cron-initiated tasks.

  Look for and remove unneeded tasks started by cron.

- Remove unnecessary process monitors.

  These run at closely scheduled intervals, consuming high numbers of CPU cycles.

- Reduce processor usage by idle Linux guests.

  Ensure that idle Linux guests do not consume unnecessary processor resources. Things to consider are:

  - If it is not the default in your Linux distribution, enable the timer patch.

    Waking the Linux scheduler 100 times per second wastes processor resources.

  - Eliminate *are you there* pings.

    Do not ping an idle guest simply to verify that it is alive.

  - Do not measure performance on idle guests.

    Measuring an idle guest costs processor resources.

- Consider infrastructure processing requirements.

  Consider the cost of various infrastructure configurations, such as:

  - Network configuration

    Network costs are discussed in Chapter 11, “Network considerations” on page 167.

  - Installation and cloning costs

    See 9.5, “Infrastructure cost” on page 136, for a comparison of the costs of installing Linux guests.

- Prioritize workloads.

  When there are processor constraints, use share options to determine what work gets done.

A wide range of performance tips can be found on the IBM developerWorks® Web site at:

`http://www.ibm.com/developerworks/linux/linux390/perf/index.html`

9.1.1 Processor performance on a constrained system

When the processor is constrained, the other resources are relatively underutilized. There is little point in tuning other subsystems when the processor is already overcommitted. When the processor is constrained, all options to reduce processor requirements should be evaluated, and appropriate action taken to remove this bottleneck. We discuss some options in 9.6, “Performance effect of virtual processors” on page 140.

9.2 The effect of idle servers on performance

In a shared resource environment, there is no room for unnecessary processes. Servers that run cron jobs for historical reasons should be redesigned.

Under z/VM, as a shared resource environment, it is not optimal to wake up servers to make sure that the servers are active, or to monitor them to query current activity. In this environment, the virtual machines are tailored for optimal performance, and having unneeded interrupts or work being performed detracts from the resources available for productive work.

In a virtual environment, one of the most expensive uses of resources is waking up an idle server simply to perform a trivial task.

Pinging a server to see whether it is alive keeps the server active. Also, simple checks to confirm whether particular processes are active can cause a large, cumulative drain on CPU. Both of these practices use resources that are likely better utilized by other servers performing real work. This type of expense should *always* be monitored and minimized.

These are two of the more common mistakes made in a virtual server infrastructure. It is generally not necessary to ping applications to ensure that they are *alive* or to monitor the performance of the virtual servers. When idle, the servers would normally not take significant amounts of either storage or processor resources. But when the applications are pinged or the servers are monitored, the server must have resident storage to provide the appropriate positive responses. A far better approach in this virtual environment for monitoring is to utilize the existing and mostly free (in terms of resource requirements) VM monitor.

By default, most Linux distributions start several services that might not be needed. To illustrate the cost of these services, we show the CPU usage and DASD I/Os in Figure 9-1.

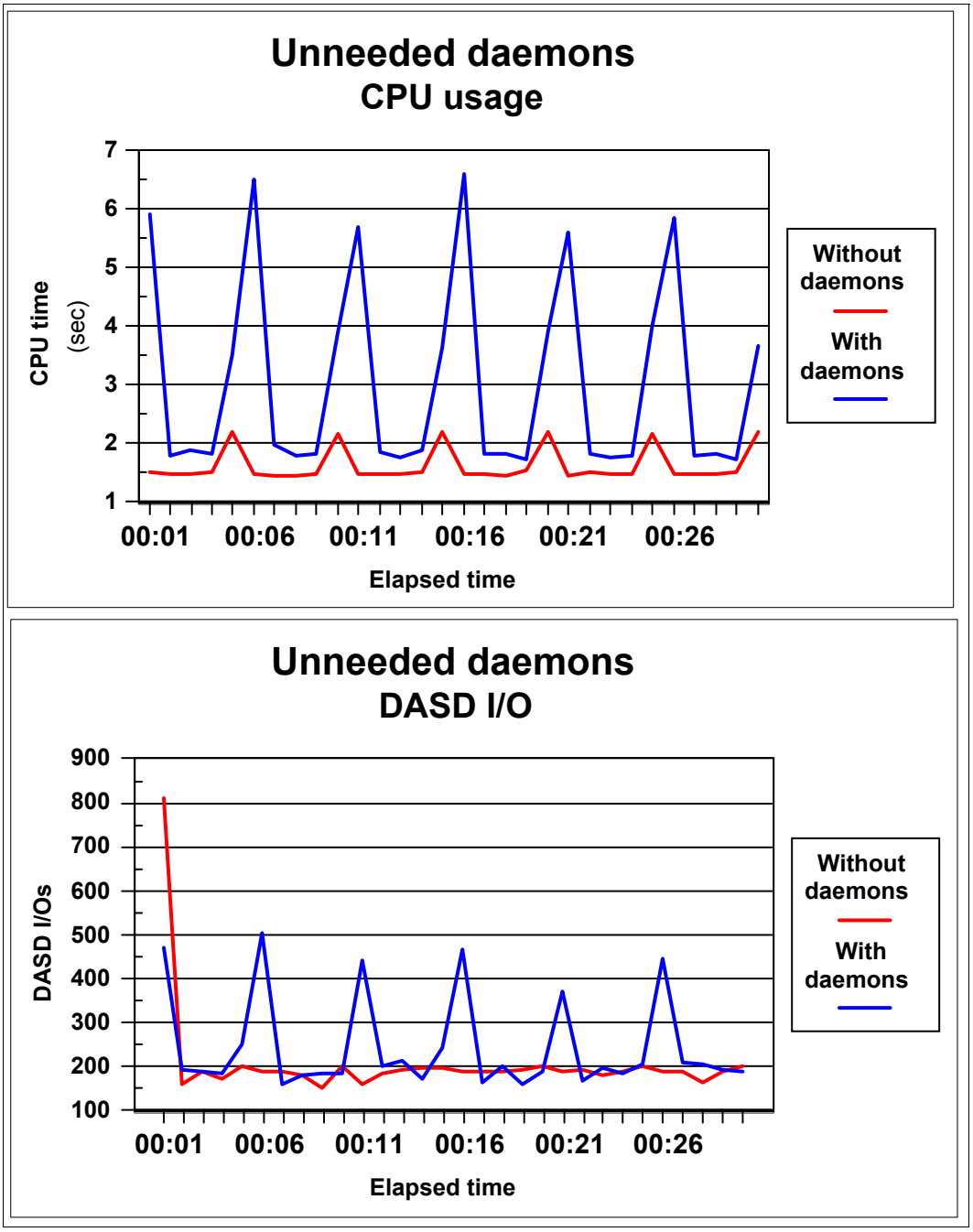


Figure 9-1 The effect of unneeded daemons on CPU and DASD usage

Services that might be considered unnecessary in a Linux guest include:

- sendmail

  If receiving or delivering mail is not required on the Linux guest, consider stopping the sendmail server.

- anacron, atd, and cron

  These daemons are responsible for initiating tasks at regular intervals. If no useful tasks are automatically scheduled, stop these services.

- autofs, nfs, nfslock, and portmap

  The autofs daemon is responsible for automatically mounting file systems. nfs and nfslock provide Network File System (NFS) support. portmap provides Remote Procedure Call (RPC) services required by NFS.

- lpd and xfs

  Printing services are provided by the lpd daemon. xfs provides X-Windows fonts.

- inetd/xinetd

  The inetd (or the replacement xinetd) daemon manages Internet services such as FTP and Telnet.

- resmgr

  The resmgrd is the resource manager daemon that allows applications to lock and access device files.

- dbus

  The dbus is the D-BUS message bus daemon and is used for application-to-application communication.

- hal

  The hal is a *hardware abstraction layer* daemon. It offers a unified method to manage and control devices like in the /dev three on your system.

9.3 The Linux timer modification

Traditionally, the Linux kernel keeps track of time using a timer that interrupts the kernel at a constant rate. On each interrupt, the global variable *jiffies* is incremented and various queues are inspected for work. On dedicated hardware, this has a minimal performance effect. However, when running Linux running on z/VM, this is not a good idea.

Although this method of keeping track of time may not be the most efficient, the biggest problem for Linux on z/VM is that the Linux guest uses a little bit of the CPU cycles every 10 ms. This causes the z/VM scheduler to keep the guest in queue. Therefore, unused memory pages in the Linux virtual machine cannot be trimmed. With many Linux guests holding on to their working sets, z/VM memory fills up quickly.

This problem was addressed with the so-called *on demand timer* patch. The patch is included in Linux on System z distributions like Novell SUSE SLES8, SLES9, SLES10, and Red Hat RHEL3 Update 1, RHEL4, RHEL5.

This patch does away with the 10 ms timer tick and sets the timer to pop only when the kernel needs to wake up. Even though this does not make a real *zero load idle Linux guest*, the periods between two timer ticks are normally long enough for z/VM to recognize the guest as idle (and start taking measures to trim memory pages if necessary).

Note: The on demand timer patch can be enabled by changing the kernel config option CONFIG_NO_IDLE_HZ=n and a kernel recompile. Alternatively, the on demand timer patch can be activated and deactivated permanently with the following convenient method. Use the **sysctl** command to switch the setting in the proc file system:

1. 'sysctl -w kernel.hz_timer=1' enables the 100 Hz timer. The on demand timer patch is deactivated.
2. 'sysctl -w kernel.hz_timer=0' disables the 100 Hz timer. The on demand timer patch is activated.

Enable & disable “on demand” timer patch on the fly (no reboot required but not a permanent change, lost after reboot) echoing values 0 or 1 into /proc/sys/kernel/hz_timer.

More details can be found following the link:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_how_timerpatch.html

Example 9-1 shows two virtual machines:

- RMHTUX01 runs a kernel with the timer patch applied.
- LNXR09 runs an un-patched kernel.

The%CPU column shows that the Linux machine with the on demand timer uses fewer CPU resources. The amount of CPU resources looks small, but we should think of many guest scenarios as the consolidation of some hundred servers that are already reality today.

Example 9-1 Two Linux guests, one with timer patch applied

```
<USERID> %CPU %CP %EM ISEC PAG  WSS  RES   UR PGES SHARE VMSIZE TYP,CHR,STAT

LNXR09    .62 .06 .56  .20 .00  28K  34K   .0    0   100   128M VUX,DSC,DISP
RMHTUX01  .01 .00 .01  .00 .00  23K  24K   .0    0   100   128M VUX,DSC,DISP
```

Even though saving CPU resources is very welcome, the significant benefit of the changed timer behavior is that z/VM despatcher is able to recognize that the virtual machine is idle. When a virtual machine does not consume resources for more than 300 ms, the z/VM scheduler assumes that a transaction has ended and that the virtual machine went into a long-term wait. It is not possible to reduce this value to a shorter test idle time with standard z/VM configuration processes. When the 300 ms test idle period is over, z/VM starts to page out some of the working sets of the idle virtual machine when main memory is constrained. When the virtual machine becomes busy again (even if only after a second or less), the page fault handling pages in necessary portions of the virtual machine’s memory. This reduces the footprint of the idle Linux virtual machine.

Note: Even with the on demand timer, an idle Linux virtual machine is still reported as active by VM because it wakes up frequently. Be careful when using the terms *idle* and *active* in this context. The idle Linux machine probably behaves more like an interactive CMS user.

When a Linux guest with the on demand timer still uses a lot of CPU time when idle, that is normally caused by some process or kernel thread requesting frequent wake-up calls. In some cases, these frequent wake-up calls are part of the application design or are caused by running daemons on the Linux system.

9.3.1 Analyzing the timer ticks

With the on demand timer installed in the kernel, the number of timer interrupts goes down. In some systems, the number of timer interrupts stays high. If the cause is not obvious, you can look at the number of timer ticks and the process requesting them.

A simple way to count the number of timer ticks is to run a TRACE EXT in the Linux virtual machine for a fixed period of time and look at the output. Example 9-2 illustrates a program executed from a privileged user ID (class C) to count timer ticks.

Example 9-2 Sample program (COUNTEXT EXEC) to count timer ticks

```
/* COUNTEXT EXEC     Trace timer interrupts to count them          */
arg uid ; if uid = '' then exit 24
cmd = 'CP SEND CP' uid

cmd 'SPOOL PRT PURGE'
'CP SLEEP 1 SEC'
cmd 'TRACE EXT 1004 PRINTER RUN'
'CP SLEEP 60 SEC'
cmd 'SPOOL PRT' userid() 'CLOSE'
'CP SLEEP 1 SEC'
cmd 'TRACE END ALL'
```

The SLEEP command in the program lets the target Linux virtual machine run for 60 seconds and closes the printer spool file after that time. We execute the script in Example 9-3.

Example 9-3 Executing COUNTEXT EXEC

```
countext rmhtux02
RDR FILE 0180 SENT FROM RMHTUX02 PRT WAS 0018 RECS 0083 CPY
```

The spool file arrival message indicates that TRACE wrote 83 records to the spool file in the one-minute interval the script was sleeping. Expect this number to vary between tests.

When the number of timer ticks is higher than expected, we trace the process requesting the wake-up calls. A good address to trace is the entry point of the schedule_timeout() function in the kernel. You can find its address in the current System.map file on your system. Alternatively, you can use /proc/ksyms, as shown in Example 9-4.

Example 9-4 Determining the address of schedule_timeout

```
# cat /proc/ksyms | grep schedule_timeout
0010ae74 schedule_timeout
```

Start the trace from the 3270 console. If you use the **hcp** command through a Telnet session, you create work on the Linux guest that obscures the measurements. The following CP TRACE command prints the value of the current entry to the schedule_timeout() function:

```
#CP TRACE I R 10AE74.2 TERM RUN CMD D C40.4
```

Example 9-5 shows the trace output.

Example 9-5 Tracing the value of current entry to schedule_timeout()

```
-> 0010AE74'  STM   908FF020 >> 005C3EB8    CC 0
V00000C40  005C4000                          06 L00000C40
 -> 0010AE74' STM   908FF020 >> 01F49E30    CC 0
V00000C40  01F4A000                          06 L00000C40
 -> 0010AE74' STM   908FF020 >> 005C3EB8    CC 0
V00000C40  005C4000                          06 L00000C40
 -> 0010AE74' STM   908FF020 >> 005C3EB8    CC 0
V00000C40  005C4000                          06 L00000C40
```

Tip: The current variable identifies the process running when the call to schedule_timeout() is made. The task_struct for the process is located 8192 (0x2000) bytes before the current. The process identifier (PID) is located at offset 0x70 into the task_struct.

Using the sort stage of *CMS Pipelines,* a simple pipe gives the breakdown per process for our one-minute interval. Using **grep** against the output of **ps -ef**, we identify the process name.

9.4 OSA-Express QDIO enhancements for Linux under z/VM

Enhancements to Linux qdio and qeth drivers can increase the efficiency of Linux servers using Gigabit Ethernet or Fast Ethernet OSA-Express running QDIO. Gigabit Ethernet throughput can increase by as much as 40%, while the VM CP cost is reduced by 50%. Novell/SLES10 systems have these qdio and qeth drivers included in the distribution.

9.5 Infrastructure cost

Some of the resources used on z/VM to run Linux virtual machines can be seen as infrastructure cost. The resources used to run these utility services are not available for use by Linux virtual machines to run business applications. This does not mean that these utility services are not necessary. However, it does mean that it is often worthwhile to review those services and see whether savings can be realized.

9.5.1 Installing new systems

There are many different ways to install Linux systems. New systems can be *cloned* from an existing *golden image*, or you can do a fresh install for each system. For a fresh install, you can IPL from disk, from tape, or from the virtual card reader. The packages can be loaded from a local disk or from a remote FTP or NFS server. This is true when installing Linux on discrete servers, as well as when you install Linux in a virtual machine (though with Linux on z/VM you have some extra options to automate the install process).

When installing Linux on discrete machines, the thing that matters most is to get the job done with minimal effort (and in the least amount of time). With Linux on System z things are very different because the virtual machines share pooled resources. The resources used for the installation of a new Linux system cannot be used by other Linux virtual machines running the business applications. Unless your z/VM system has plenty of unused resources, you probably should also concentrate on doing the install using the least amount of resources.

We compared five ways to install a new Linux system on z/VM:

- RDR + FTP + Router

  The virtual machine is connected through IUCV to a VM TCP/IP stack as the virtual router. It IPLs from the virtual reader and installs the RPM packages through FTP.

- RDR + FTP

  Similar to the first method, but in this case, the Linux virtual machine has its own OSA device. This avoids the cost in the VM TCP/IP virtual router.

- QuickStart

  A single (R/O) minidisk is used that holds the starter system and a copy of the RPM packages. The minidisk is IPLed to get the ramdisk system (instead of IPL from virtual reader). The minidisk is then mounted in the ramdisk system to install the packages (which avoids the network traffic and FTP server cost).

- Breeder

  A separate Linux virtual machine (the Breeder) is used. The Breeder does a R/W link to the target minidisks of the new system and makes a copy of a preinstalled system onto the target minidisks. After the file system is copied, the *personalization* is done (host name and IP address, for example). This method also avoids the unzipping of the RPM packages.

- GUI + FTP + Router

  SuSE SLES 10 includes a working X-Windows version of YaST. Even though this is a different installer and kernel version, we include it in the measurements to show some of the differences between the install methods.

Note: The QuickStart install method should not be confused with Red Hat Kickstart. Kickstart automates installation by obtaining installation parameters from a configuration file. No user prompting is required. The biggest savings, if any, are in elapsed time for the install, and therefore, also in the memory usage. For SuSE, there are options such as Auto-YaST and Alice, but the versions we have seen still lacked some of the install options that are essential for Linux on System z.

For each of the four tests, we installed the same minimal set of packages, just enough to get a working Linux system. The 130 packages in this set are approximately 140 MB worth of RPM files to be loaded. When unpacked during the install, this results in 200 MB worth of data on the root device.

The Breeder install process is a home-grown installation process that we used. Even though the numbers are not immediately applicable to your own installation, we believe it is a representative measurement for what people are doing with various cloning approaches and DDR copies of minidisks.

Elapsed time comparison

In Figure 9-2, we show the elapsed time for the five installation methods. Because each install process uses some manual steps (navigation though YaST panels), this is not easy to measure. To get an idea of the elapsed time, we recorded the CPU usage per minute and discarded the intervals where the Linux system used little more than the idle load.

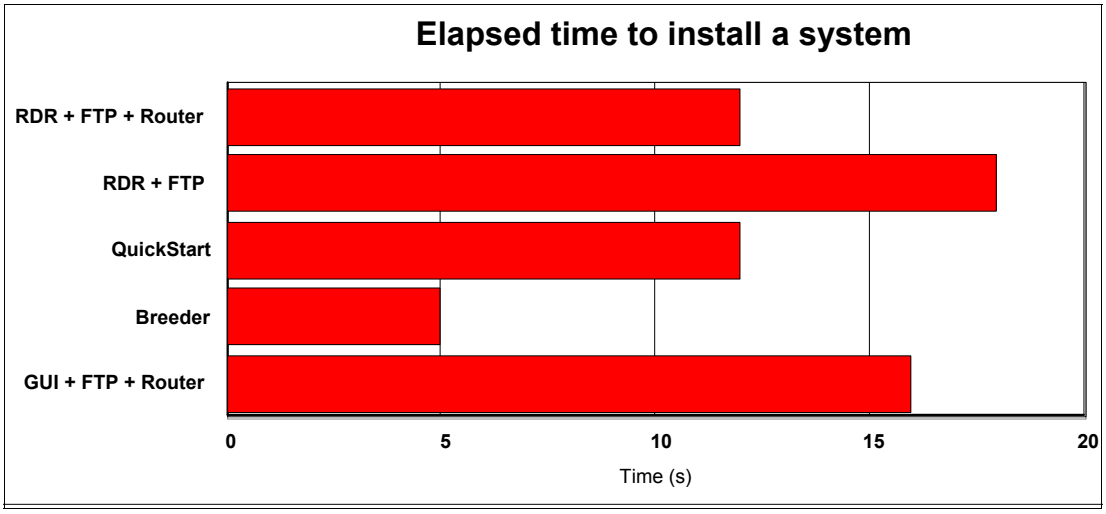


Figure 9-2 Elapsed time for installation using different methods

The Breeder method is clearly the fastest. This should not be a big surprise because it avoids some steps that are known to be relatively slow on System z. It is a bit surprising to see that taking out the virtual router does not make the second method faster. The reason for this is that the VM TCP/IP stack is more efficient in driving the OSA device (an OSA-2 Token Ring in this case) than the Linux LCS driver. We believe that this difference can be attributed to the use of Diagnose98 in the VM TCP/IP stack.

CPU time comparison

Probably more important than the elapsed time is the CPU time used for the installation. The total CPU cost, as well as the breakdown of the cost per virtual machine, is shown in Figure 9-3.

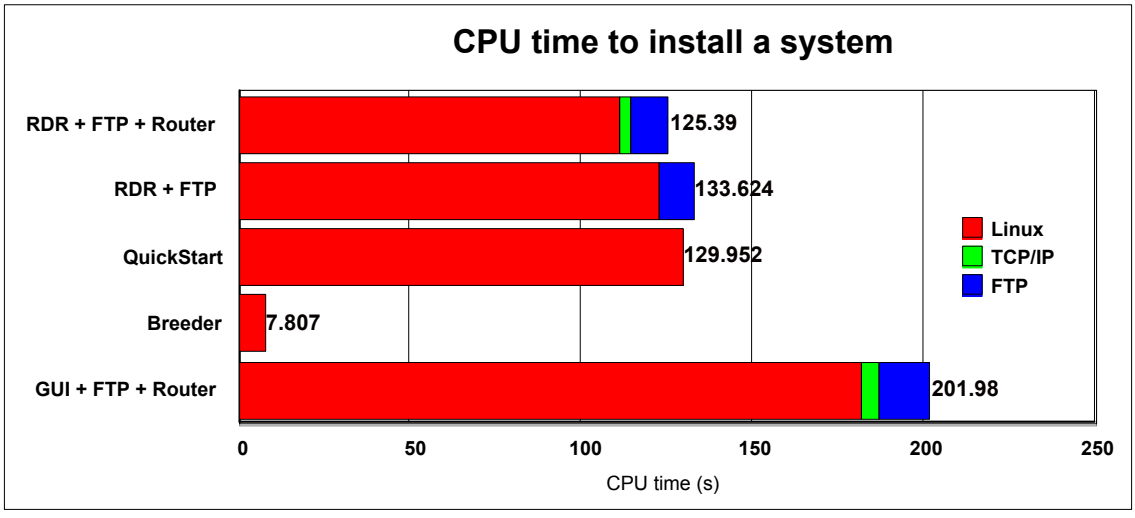


Figure 9-3 CPU time used to install a system

The Breeder method uses less CPU cycles because it avoids the cost of decompressing the RPM packages. The differences between the other methods are less obvious. For the installs that use an FTP server, the cost of the FTP server are roughly the same. The main difference between the first two methods must be attributed to the more efficient LCS device driver in VM TCP/IP. The GUI method of installation uses a lot more CPU cycles because of the X-Windows applications and the additional network traffic (the CPU portion for TCP/IP is slightly larger).

If the installation was done on a more CPU-constrained system, the larger demand for CPU cycles would have immediately translated into longer elapsed time as well.

DASD I/O comparison

Another cost item to look at is DASD I/O. Because the QuickStart method copies the RPM packages from disk instead of receiving them through the network, we expect a larger DASD I/O rate (but far less than double because RPM packages are compressed). This is confirmed by the graph in Figure 9-4.

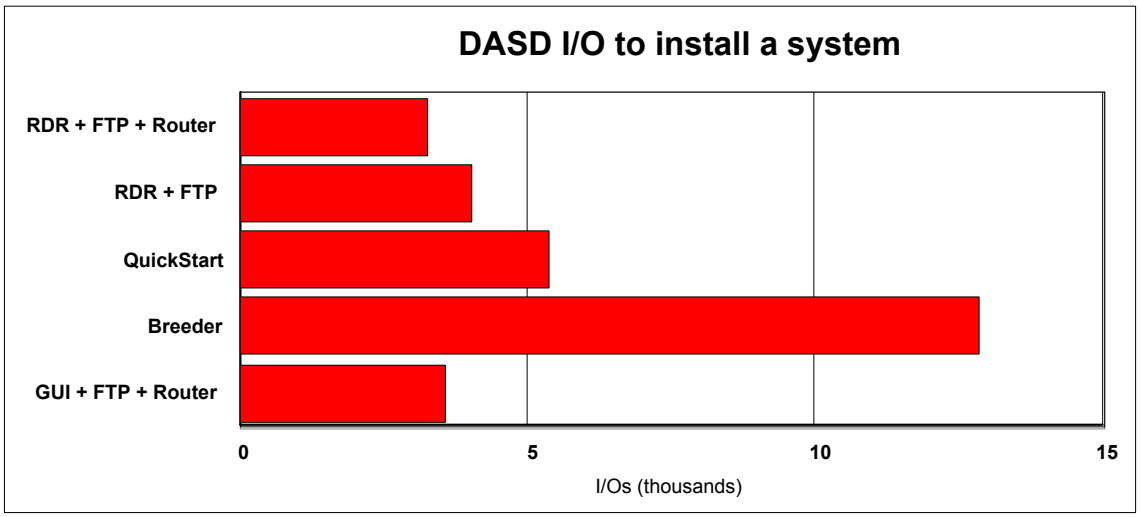


Figure 9-4 DASD I/O to install a system

It is not clear whether the difference between the two FTP methods is significant, but it is not unlikely that a Linux program will do more I/O for the same work when it takes longer to complete.

The increased amount of DASD I/O for the Breeder installation is significant. The Breeder uses an uncompressed copy of the file system to install on the target disks (this saves a lot of CPU time). The drawback of that approach is that we do more I/O while saving CPU cycles. This is a very obvious example of the trade-off you make when tuning a system. When you need to install a lot of systems in a short time, the z/VM minidisk cache does you a lot of good. It might even be attractive to make the Breeder Linux virtual machine big enough to hold the entire image of the target disk in buffer cache.

Note: Because the Breeder first copies the target disk and then applies the personalization, it is possible to build a stock of copied target disks. The only thing left would be personalization after the host name and IP address are known. The stock could be replenished during the night or some other time when sufficient resources are available to do so. It would also be possible to have a CMS service machine combine the formatting with copying the image.

Memory usage comparison

The final cost factor for installing Linux systems is memory utilization. While we do not have full numbers on the working set of the virtual machines during the install process, for Linux virtual machines, it is normally good enough to multiply the virtual machine size with the elapsed time. For the ramdisk installation system, the virtual machine size needed is approximately 128 MB.

For the graph in Figure 9-5, we included part of the memory usage of the FTP server and the TCP/IP stack where applicable

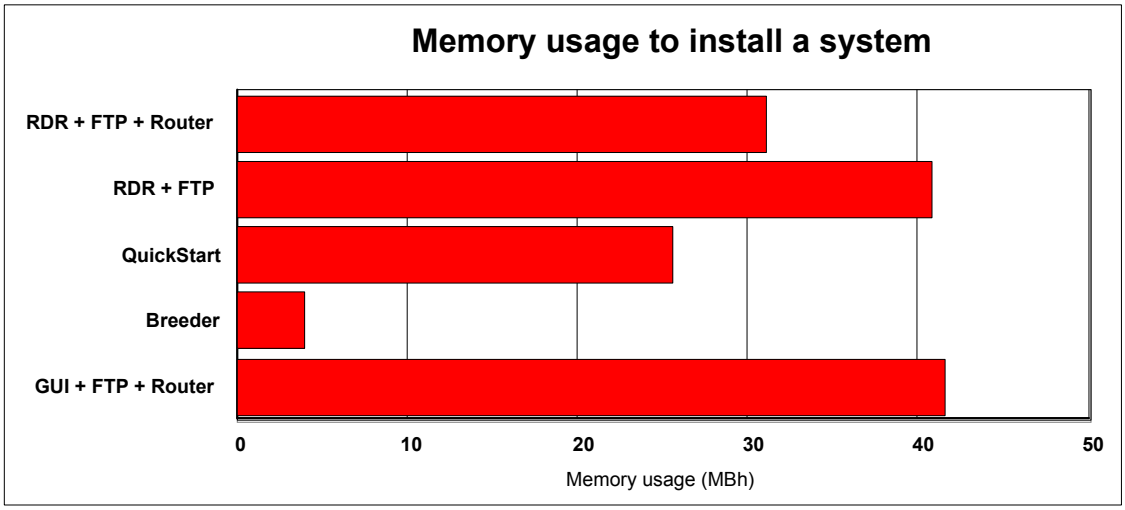


Figure 9-5 Memory usage for installation of a system

Conclusion

Considerable savings can be made by using an efficient installation process. This is an important issue if you create a lot of Linux systems or when you want to deliver new systems very quickly.

9.6 Performance effect of virtual processors

Assigning virtual processors to a Linux virtual machine can be an effective means of matching System z resources to anticipated workload. Overall throughput can be affected by the number of processors assigned to a Linux virtual machine.

9.6.1 Assigning virtual processors to a Linux guest

CP shares all real processors defined to an z/VM LPAR. All virtual machines appear to have at least one processor (referred to as the base virtual processor). Additional virtual processors can be added to a virtual machine using the CP DEFINE CPU command. As shown in Figure 9-6, virtual processors share real processor resources.

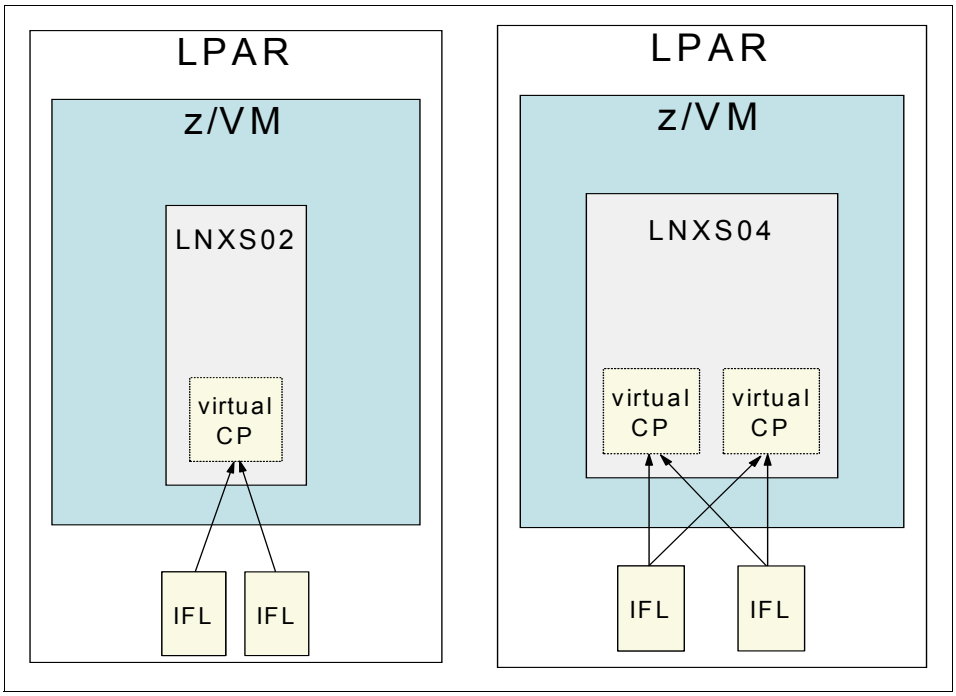


Figure 9-6 Virtual processors in a virtual machine

In Figure 9-6, Linux guest LNXS02 has access to two physical IFLs. However, because it is a virtual uniprocessor, only one IFL may run at a time. Linux guest LNXS04 is defined to have two virtual processors. Each virtual processor utilizes the physical IFLs in the LPAR.

9.6.2 Measuring the effect of virtual processors

Adding virtual processors to a virtual machine can improve performance for processor-constrained workloads. To examine the effects of defining virtual processors to a Linux guest, we consider the WebSphere Performance Benchmark Sample workload. Using the WebSphere Performance Benchmark Sample in a three-tier configuration, we determine the relative cost in terms of processor resources for each component of the workload:

- The WebSphere Application Server
- The HTTP server
- The DB2 server

Note: We run each component in its own virtual machine and measure the processor resources expended during a simulation running five concurrent clients. The processor time spent by each virtual machine is attributed to the WebSphere Performance Benchmark Sample component running in the respective z/VM guest.

In Figure 9-7, we examine the effect that varying the number of virtual processors has on CPU utilization. In this scenario, two IFLs are dedicated to the z/VM LPAR. The Linux guests running the IBM HTTP and DB2 servers both run with a single virtual processor (the default). The number of virtual processors allocated to the WebSphere guest is varied from one up to four.

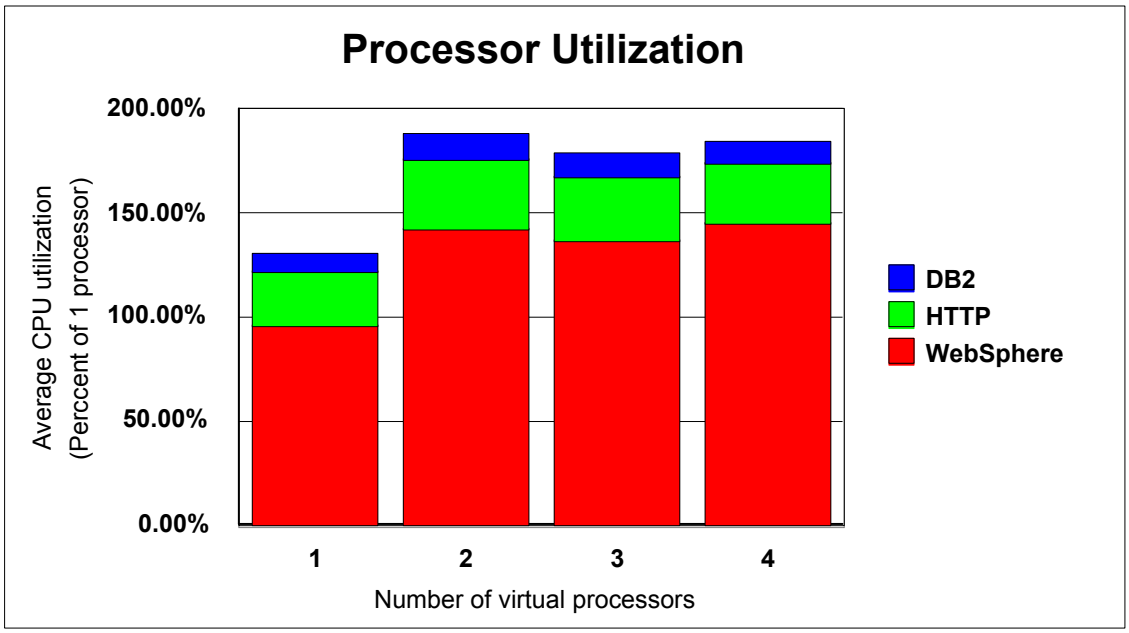


Figure 9-7 Measuring the effect of virtual processors on CPU utilization

From Figure 9-7, we see that this workload is processor constrained. With one virtual processor, the guest running WebSphere consumes more than 95% of a single real processor. When two virtual processors are allocated to the guest, processor utilization increases to 143%. Note that increasing the number of virtual processors beyond the number of real processors does not increase the CPU utilization for the WebSphere guest.

Using the average response time and average CPU time utilization, we derive the cost of adding virtual processors. Using the reported transaction rate and measured average CPU time expended in each Linux guest, we calculate the average cost of a WebSphere Performance Benchmark Sample transaction (measured in milliseconds per transaction) in Figure 9-8.

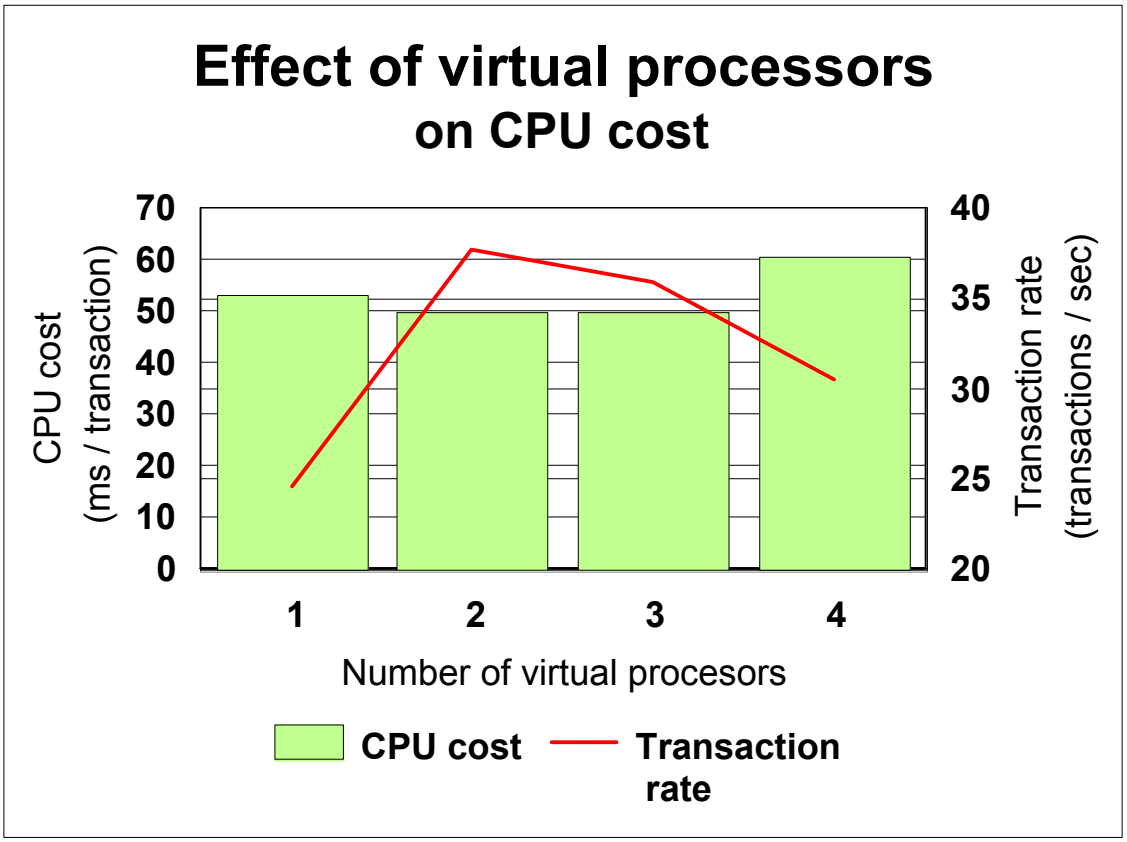


Figure 9-8 Measuring the cost of adding virtual processors

In Figure 9-8 we see that when the number of virtual processors matches the number of real processors, the average transaction cost decreases slightly. More importantly, the average transaction rate increases significantly. Note, however, that as the number of virtual processors is increased beyond the number of real processors (two in this case), the overall transaction rate decreases, and the cost per transaction increases. This is due to the additional scheduling overhead incurred by CP.

When a Linux guest runs a processor-constrained workload, we recommend:

- Defining the same number of virtual processors to the guest virtual machine as the number of real processors available to the LPAR.
- Never defining more virtual processors to the guest virtual machine than the number of real processors available to the LPAR.

10

Tuning disk performance for z/VM Linux guests

This chapter describes Direct Access Storage Device (DASD) and SCSI disk tuning for z/VM Linux guests. Topics include:

- Disk and access options
- Disk setup in z/VM and Linux on System z
- Factors that influence disk I/O
- Maximizing the operation of disks

© Copyright IBM Corp. 2003, 2008. All rights reserved.

10.1 Disk options for Linux on System z

There are many options to choose for disk devices. There are Direct Access Storage devices (DASD) and Small Computer System Interface (SCSI) disks, and z/VM controlled devices like minidisk. Some of the devices are able to use different disciplines and access methods to communicate.

10.1.1 DASD with ECKD or DIAGNOSE

Linux on System z supports DASD devices like 3390 Mod. 3, 3390 Mod.9, and 3390 Mod 27. Today these devices are emulated by storage subsystems. Different methods to connect to the DASD devices are available to choose from. The relatively slow ESCON® channel is still supported, but faster FICON adapters are available. Recent distributions come with the DIAGNOSE discipline to access ECKD. The latest distributions offer the DIAGNOSE access method for 64-bit Linux on System z (z/VM 5.2 or later is required).

10.1.2 SCSI disk

Support of FCP enables System z servers to attach to SCSI devices as well as access these devices from z/VM or Linux on System z. This connectivity provides enterprises with a wide range of choices for storage solutions and may allow use of existing storage devices. Fibre Channel (FCP Channel) support means that Linux instances on System z can attach to and communicate with SCSI devices using the Fibre Channel Protocol (FCP).

10.1.3 z/VM minidisks

This is the standard and most basic way that z/VM allocates disk resources to users and guest systems such as Linux. A large system volume, such as an emulated 3390-3, contains 3339 *cylinders* when formatted before being subdivided, of which 3338 are available for allocation to users. (The first cylinder, numbered 0, is reserved). Firstly, a new volume must be prepared by formatting, and then allocating the user extent, giving it a type of PERM, denoting permanent user space. These prepared 3338 cylinders are typically split into portions of varying length and allocated to users, often by using the DIRMAINT tool. An entry is created in the user VM Directory entry to define the minidisk (MDISK). If a directory management tool is not available, this entry can be prepared using the VM Xedit editor. A line defining a minidisk in the Directory looks like this:

```
MDISK 0191 3390 61 20 LX6U1R MR
```

This shows, reading the fields from left to right, that this line entry is defining a MDISK, the virtual address, the emulated disk device type, the starting cylinder address, the size in cylinders, the volume name, and the mode or level of disk access privilege. Here MR denotes a conditional multiple-write mode.

10.2 Factors that influence disk I/O

In this section we discuss factors that influence disk I/O.

DASD I/O

When evaluating DASD I/O performance, overall response time consists of several components, as depicted in Figure 10-1.

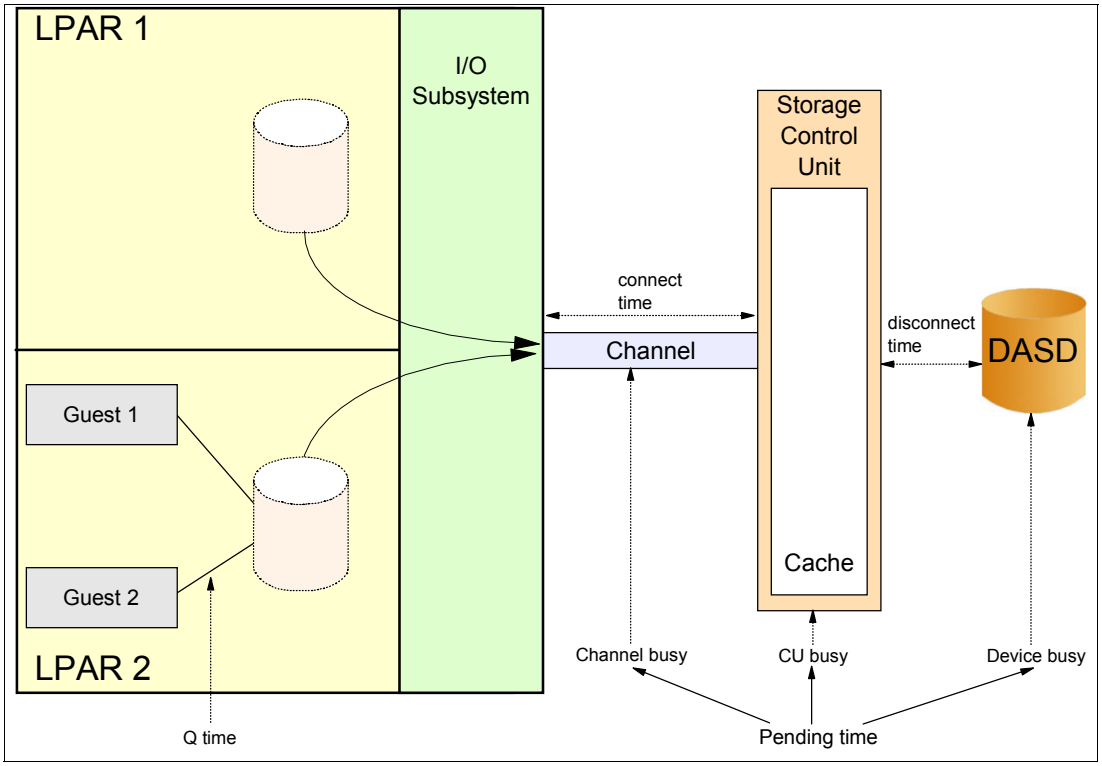


Figure 10-1 Components of overall DASD response time

These components are:

- Queue time

  Queue time is a result of multiple users simultaneously accessing a device. If the device is busy servicing an I/O request, additional I/O requests wait in the queue. The length of time an I/O request waits is the queue time. This component is the most variable, and under heavy load, can be the most serious. High queue times can indicate operating system contention, or a high device service time (as I/O requests take longer to complete, requests for service from other users can cause queuing for heavily used devices).

- Connect time

  Connect time is the actual time to transfer the data on the channel, normally less than 2 ms for a 4 K block of data.

- Pending time

  Pending time is the time required to start an I/O operation. Normally, this amounts to less than 1 ms. High pending time reflects contention on the I/O path. This might be due to contention on the channel, on the control unit, or on the DASD device.

- Disconnect time

  Disconnect time is the time required by the control unit to access the data. This includes rotational delays, seek delays, cache management, and processing time inside the control unit. Disconnect time is normally less than 2 to 3 ms on cache controllers.

- Service time

  Service time is the sum of pending, connect, and disconnect times.

- Response time

  Response time is the sum of queue and service times.

SCSI I/O

When evaluating SCSI I/O performance, overall response time consists of several components:

- Queue time

  If the device is busy servicing an I/O request, additional I/O requests wait in the queue. The length of time an I/O request waits for queuing is the queue time. This component is the most variable, and under heavy load can be the most serious. High queue times can indicate operating system contention, or a high fabric latency (as I/O requests take longer to complete, requests for service from other users can cause queuing for heavily used devices).

- Fabric latency

  This is the time from entering the fabric until coming back. This includes all the time that is needed on the storage subsystem and in switches.

- Channel latency

  Additional time spent in the channel to fulfill the request.

10.3 Observing disk performance

We monitor performance and identify performance issues using resource management tools. There are many ways to collect the data. Besides the online tools there are ECKD and SCSI statistics, too.

10.3.1 DASD statistics

Linux can collect performance stats on DASD activity as seen by Linux. The data collection is carried out by the DASD driver.

Example 10-1 DASD statistics - how to use

```
Turn on with
echo on > /proc/dasd/statistics
Turn off with
echo off > /proc/dasd/statistics
To reset: turn off and then turn on again
Can be read for the whole system by cat /proc/dasd/statistics
Can be read for individual DASDs by tunedasd -P /dev/dasda
```

For more details refer to the following Web site:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_how_tools_dasd.html

10.3.2 SCSI statistics

The Linux kernel 2.4 version of the device driver for FCP provides some basic statistics of the I/O traffic pattern for SCSI devices for Linux on System z.

For Novell/SUSE kernel Version 2.4.21-251 and later the device driver gathers independent statistics for each FCP subchannel. Statistics are turned off by default and may be activated as described below.

More details can be found at:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_how_tools_fcp.html

10.3.3 Monitoring disk statistics using the z/VM tools

Figure10-2 shows some of statistics that are available for each DASD volume. The criticality of tuning at this level is much lower than in the past, as the more modern disk hardware now has its own sophisticated management interface. But when making choices about whether to implement features such as minidisk cache, which are active much further up the I/O chain, and closer to the application, it is necessary to analyze and understand the existing access patterns, for example, relative numbers of reads and writes, at the device level. Device level contention is typically much lower on a well architected system nowadays, but if problems do arise, or you are seeking to achieve the highest level of optimization, these are examples of the statistics that would prove useful. The listing shown is from the IBM PAF. Disk statistics are also available in the VM Performance Toolkit and Tivoli Omegamon tools.

```
2600.Resp                          Response time, Volume 1LI900 (Milliseconds pe
2600.Qlen                          Queue length, Volume 1LI900 (SSCHs queued)
2600.Serv                          Service time, Volume 1LI900 (Milliseconds per
2600.Util                          Device utilization, Volume 1LI900 (Percent of
2600.IOrate                        SSCH and RSCH rate, Volume 1LI900 (Number per
2600.Blocks                        4096 byte blocks, Volume 1LI900 (Number per s
2600.DiscBlock                     Disconnect time , Volume 1LI900 (Milliseconds
2600.MDCssch                       SSCH satisfied via MiniDiskCache , Volume 1LI
2600.Writes                        Write operations  , Volume 1LI900 (Number per
2600.Reads                         Read  operations  , Volume 1LI900 (Number per
2600.IOrate                        .reads .writes .MDCssch
2600.IO_Block                      4096 byte blocks, Volume 1LI900 (Number per I
2600.IO_Conn                       Connect state, Volume 1LI900 (Milliseconds pe
2600.IO_Disc                       Disconnect state, Volume 1LI900 (Milliseconds
2600.IO_Pend                       Pending state, Volume 1LI900 (Milliseconds pe
2600.IO_Queue                      Queued time , Volume 1LI900 (Milliseconds per
2600.IO_QueueQF                    Requests queued , Volume 1LI900 (Number) EXEC
2600.IO_Resp                       Requestor waiting time, Volume 1LI900 (Millis
2600.IO_Resp              *set*    IO_Conn IO_Disc IO_Pend IO_Queue
```

Figure 10-2 Short extract showing disk information and chart titles included in a postprocessed file

In Figure 10-2 note that many more low level details are available for this and each other volume.

The chart shown in Figure 10-3 is an example that shows the response time and the overall I/O rate for a DASD volume. Average response time is just over one millisecond per I/O, which is relatively good for this system. The average I/O rate is 130.9 I/O per second. The output is from the PAF tool.

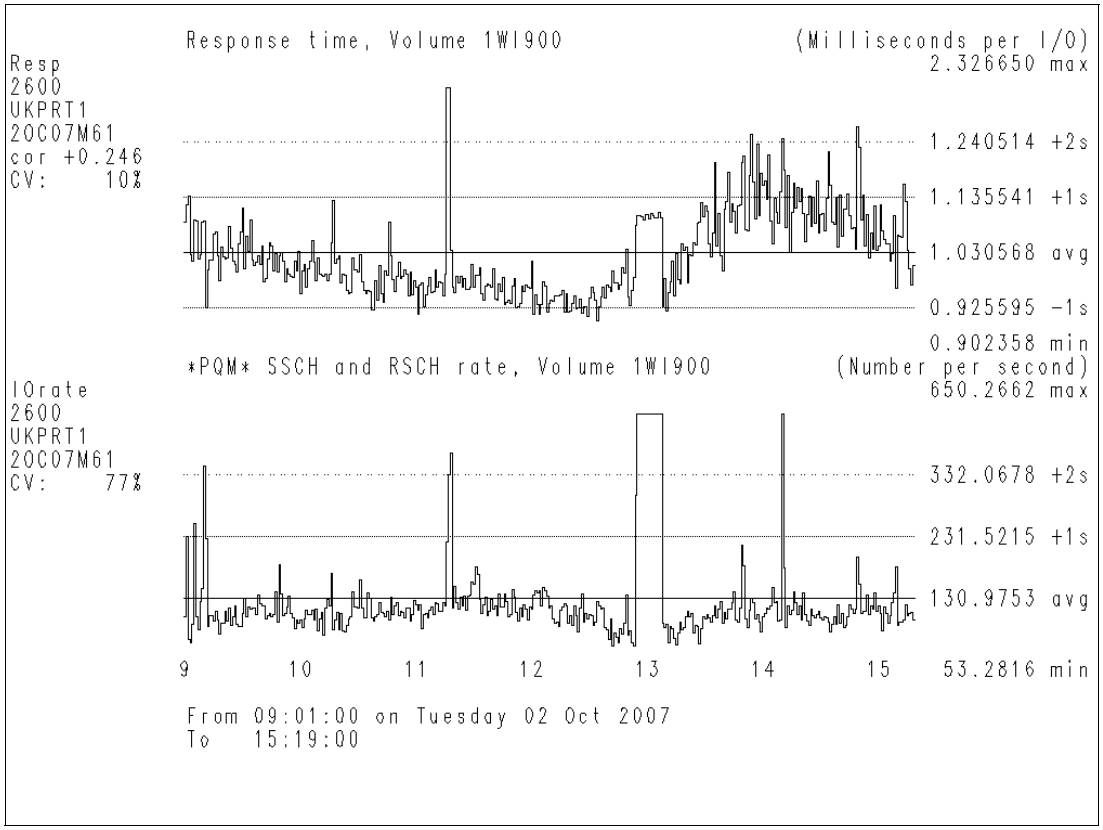


Figure 10-3 Response time and I/O rate chart for a single volume

The chart in Figure 10-4 shows a breakdown of the overall I/O response time for the same volume, showing the pending time, and the disconnect and connect time, as discussed in Figure 10-3 on page 150. Note that the pending and disconnect times are low in absolute terms, and relative to the connect time, the majority of the response time is spent transferring data to the disk, with no queue time. This output is also from the PAF tool.

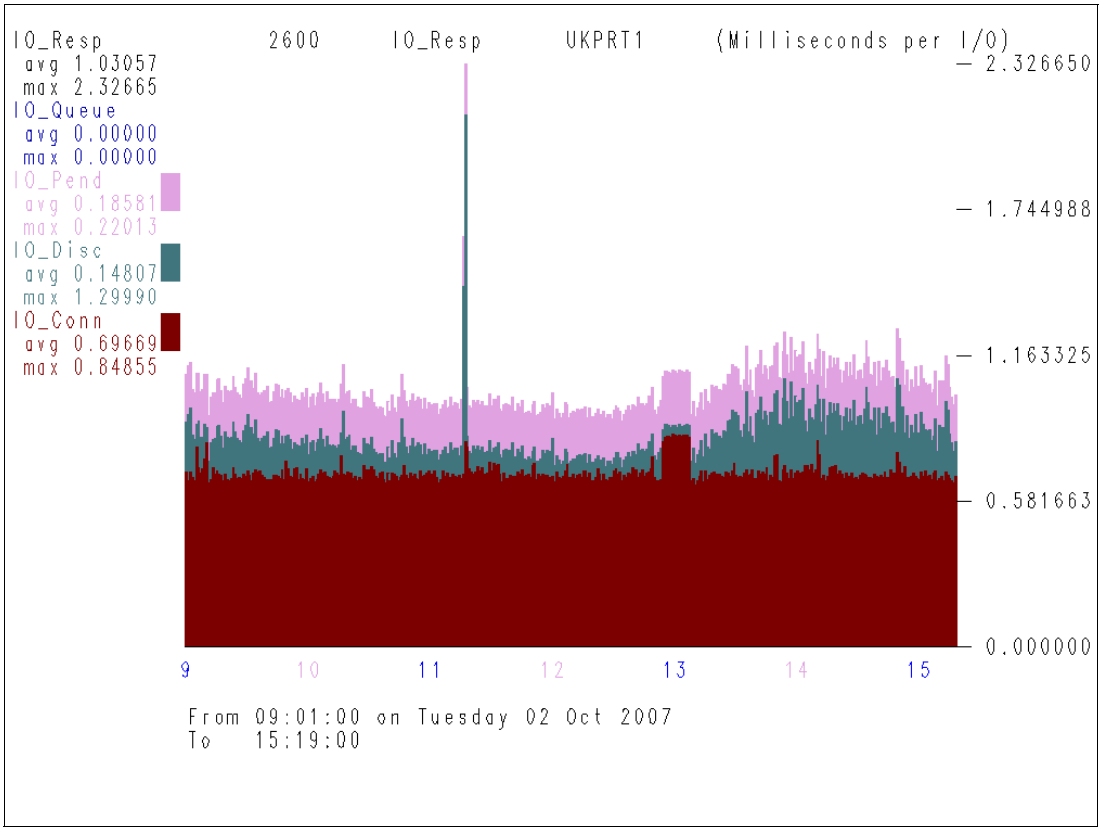


Figure 10-4 Overall I/O response time breakdown chart

10.4 Maximizing the operation of disks

In this chapter we give recommendations about how to tune disk devices for optimal performance.

10.4.1 I/O recommendations for DASD

In general, to optimize I/O performance, use more parallelism when possible.

Use more disks

I/O operations to a single physical disk are serialized. Parallel simultaneous I/O operations are faster than I/O operations to a single disk.

Reduce contention points

Contention for I/O resources can occur at several points in the I/O path:

- DASD contention

  Multiple virtual machines can experience DASD contention when simultaneously accessing data on the same DASD device.

- Control unit contention

  Contention can occur when two or more DASD devices share a common control unit.

- Channel contention

  Contention can occur during simultaneous access of the control unit or DASD devices sharing the same channel.

10.4.2 I/O recommendations for SCSI

When using SCSI disks, keep them equally alternating over FCP adapters and WWPNs. As with DASD, reduce contention points and use disks that are located physically in different ranks of the storage server. Do not use consecutive disk addresses, but alternate the clusters, logical control units, wwpns, device adapters, and ranks.

10.4.3 I/O recommendations for z/VM minidisks and Minidisk cache

When using minidisk allocations for a Linux guest, the usual placement considerations apply just as they would to a non-Linux disk. You should aim to stripe the DASD allocation for the system to engage as many channels, chpids, and hardware-level caching as possible. Then spread the minidisks so as to ensure that there is a balance in terms of overall I/O rates to all of the DASD volumes allocated. Do not place critical disks on the same volume, and ensure that VM paging, spooling, system residency volumes, online directory areas, and Linux swap disks are not on DASD volumes shared by important, performance-sensitive Linux application minidisks. These system-level infrastructure volumes are best placed apart from Linux and other application volumes, and they should not share volumes with each other.

Minidisk cache (MDC) is a data in memory (DIM) write through caching technique that can, according to the applications and usage patterns, cut the amount of real I/O that is done to DASD very dramatically. Values of 50% less DASD I/O can be achieved where sufficient main storage and expanded storage memory resources for MDC are available. (But note that the use of expanded storage for MDC is far less effective than using main storage.) There may be some trade off with increased paging, if the amounts of memory devoted to minidisk cache cause memory constraint to become significant. For a Linux minidisk to benefit from MDC, you need to understand the usage patterns, principally the number of reads and writes that are being performed. There is a processor overhead when updating MDC when the read operations are carried out. Unless the Linux minidisk is mainly read only, or has a large majority of reads compared to writes, you should consider running with MDC off for that disk, as the potential benefit from saving real I/O may be wiped out by the processor overhead.

Note: In any event, you should not use MDC for Linux swap disks.

10.4.4 Performance capabilities of hardware options

Enterprise Systems Connection, ESCON, and the newer Fibre Connection, FICON, provide I/O connectivity based on fiber optic technology. FICON is faster than ESCON. FICON is available in various versions that offer different bandwidth.

SCSI uses connections via switch and uses the Fibre Channel Protocol (FCP).

Comparison ECDKD DASD versus SCSI over FCP

Traditional System z I/O uses the channel subsystem and issues CCWs to access ECXD, CKD, or FBA disks. In an operating system like Linux, this requires a transformation from the block-oriented block device layer to CCW chains. From the ECKD protocol they do not use the count and key functions. The channel subsystem transfers the CCWs to the storage subsystem. The storage subsystem works with standard SCSI disks, and there the channel programs have to formate back to fix block format. See Figure 10-5.

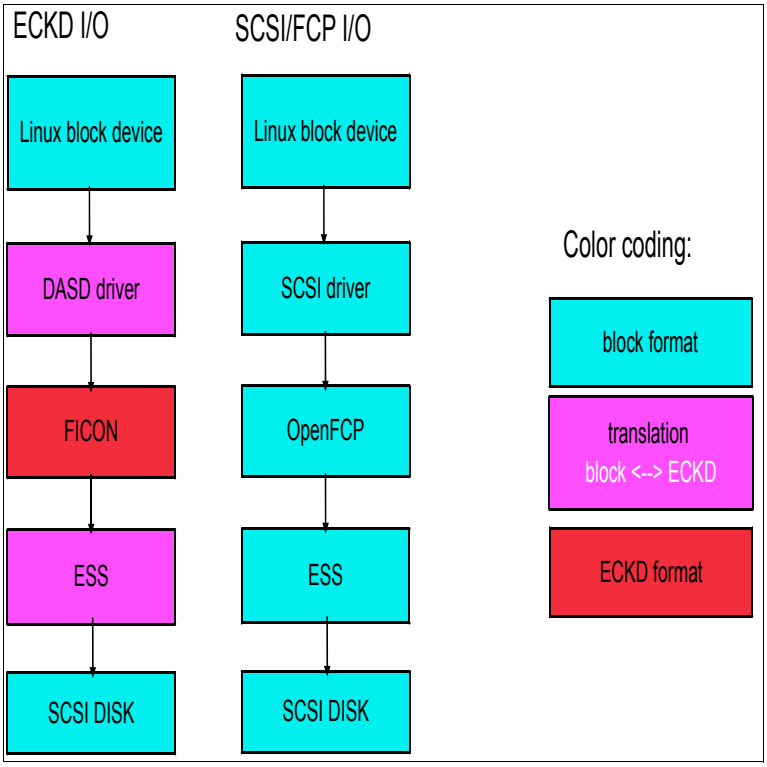


Figure 10-5 Disk access translation

Information about performance of SCSI and ECKD disk may be obtained here:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_res_dasd_ds8000_ECKD.html

Generations of storage subsystems

The biggest changes in performance are observed when migrating from one generation of storage subsystem to the next generation (choosing a post model). The caches usually are bigger and the performance of the storage subsystem increases tremendously.

Note: Keep in mind that shorter run times have a side effect. With the increase of the throughput we also need more CPU to drive the I/Os. This is true when we migrate from an older storage server to a newer one or when we replace 1 or 2 Gbit channels with 4 Gbit channels or when we use SCSI disks instead of ECKD to get more throughput.

10.4.5 DASD block size

The formatting tool *dasdfmt* provides options to format DASD devices with different hard block sizes. The dasdfmt block size of 4096b shows the best results in IOZone benchmark throughput and free disk space after formatting the DASD device. Further tests showed that this statement is independent to the request size issued by the application. Even small application request sizes (that is, 512b) showed the best throughput with 4096b dasdfmt block size.

10.4.6 IBM storage subsystems caching modes

Caching modes are strategies used by the storage subsystem to optimize its cache hit rate.

ESS caching modes

The Linux FCP driver uses the default ESS subsystem operating mode *normal cache replacement*. The mode cannot be changed for FCP/SCSI disks. This mode enables the ESS to switch between the standard algorithms depending on the actual I/O pattern. This is the preferred setting, also for DASD ECKD disks.

The subsystem operating mode can be set for every single DASD individually and lasts until the next reboot of the Linux system. After a reboot the operating mode reverts to the default. If you want to set the mode permanently for a specific DASD, include the mode switch in a Linux startup script (for example, by adding the commands to the appropriate run level scripts in '/etc/init.d/rc3.d/').

Define the caching behavior on the storage server. The following caching modes are supported by modern Enterprise Storage Servers (ESSs):

- Normal (normal cache replacement)

  We recommend this mode. It has been the default mode for some years now, starting from Novell SLES 8 updates and Red Hat RHEL4. The cache is managed using the ESS standard algorithms. This is the default value for FCP/SCSI and the default value for ECKD DASD in Novell/SUSE SLES9 and Novell/SUSE SLES8, starting with Service Pack 3 + Security update for Linux kernel, Release 20040112.

- Bypass (bypass cache)

  Cache is not used. This is a special mode and should be used carefully. Performance is moderate for random I/O.

- Inhibit (inhibit cache loading)

  Tracks added into cache are placed in the middle of the last recently used (LRU) list, which causes an early removal of the item from cache. This is a special mode that should be used carefully. Performance is good for random I/O.

- Sequential (sequential access)

  Tracks following the accessed track are added to the cache, and tracks preceding the accessed track may be removed from the cache. This was the default value (access) for Linux ECKD DASD drives in Novell/SUSE SLES7 and earlier versions of SLES8. This results in poor performance when random I/O is done.

- Prestage (sequential prestage)

  Tracks following the accessed track are added to the cache in anticipation of a near term requirement, and preceding tracks are removed from the cache.

- Record (record access)

  Only the records accessed by the channel program are added into the cache. This is a special mode and should be used carefully. Performance is good for random I/O.

DS8000 and DS6000 caching modes

A performance enhancer is the self-learning cache algorithm. The DS8000™ series caching technology improves cache efficiency and enhances cache hit ratios. The algorithm used is called Sequential Prefetching in Adaptive Replacement Cache (SARC).

SARC provides the following:

- Sophisticated, patented algorithms to determine what data should be stored in cache based upon the recent access and frequency needs of the hosts
- Pre-fetching, which anticipates data prior to a host request and loads it into cache
- Self-learning algorithms to adaptively and dynamically learn what data should be stored in cache based upon the frequency needs of the hosts

For more detailed information refer to *IBM TotalStorage DS8000 Series: Performance Monitoring and Tuning*, SG24-7146.

Tunedasd command

Use the Linux command **tunedasd** to switch between the subsystem operating modes. As the command name implies, it is working with DASD, not SCSI disks.

- tunedasd -g or --get_cache

  Get the current storage server caching behavior.

- tunedasd -c or --cache <behavior>

  Set the storage server caching behavior.

Example 10-2 Set ESS caching mode from normal to sequential

```
# tunedasd -g /dev/dasdc
normal (2 cyl)

# tunedasd -c sequential -n 2 /dev/dasdc
Setting cache mode for device </dev/dasdc>...
Done.

# tunedasd -g /dev/dasdc
sequential (2 cyl)
```

Figure 10-6 shows an example database's performance measurements. The test was carried out by the IBM Linux performance team in Boeblingen, Germany.

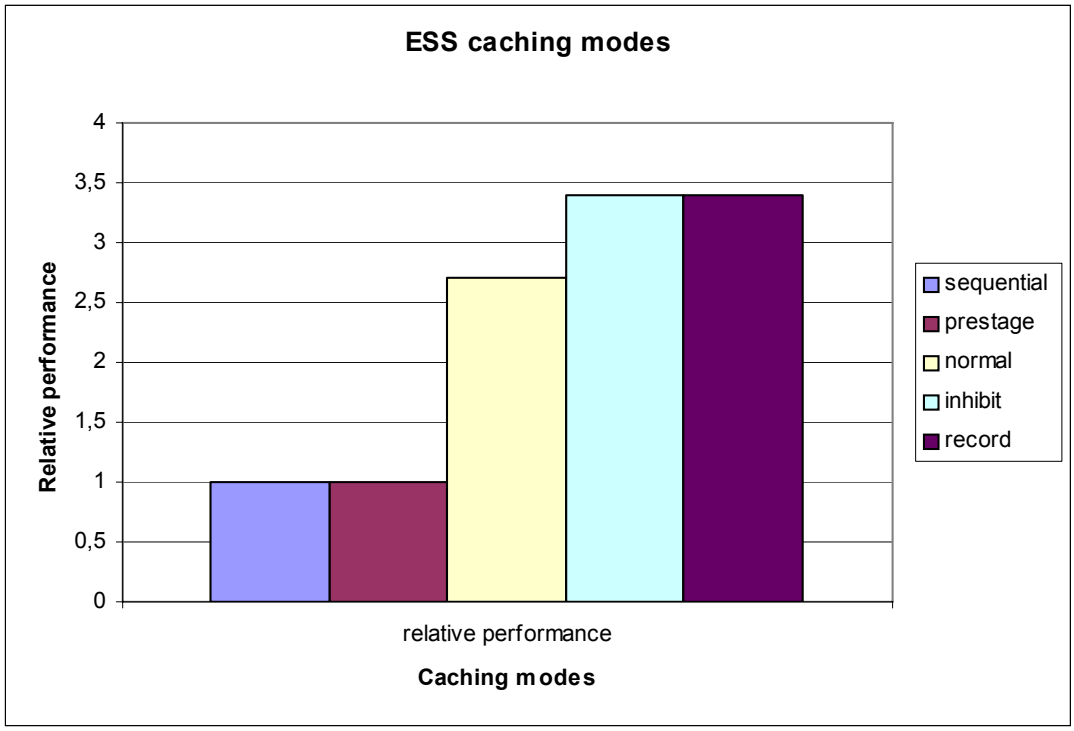


Figure 10-6 Relative performance of a database experiment using randomized access

The chart shows the influence of the caching mode on the performance. In this example the access was randomized, so random and inhibit caching modes have the best results. The normal caching mode is adapting to the randomized I/O and has a median performance. It is expect to get better performance over time if the usage pattern does not change much.

In case of sequential access to the data, the sequential caching mode has the best results. The normal mode adapts over time.

It is important to know before specifying a cache mode which access is used. Otherwise, the self-adapting and optimizing modes should be preferred.

10.4.7 Linux Logical Volume Manager

With Logical Volume Manager (LVM) data striping and software RAID emulation (RAID 0), a single logical disk is mapped to several physical disks. This allows the system to initiate up to as many simultaneous I/O operations as physical disks. The number of parallel I/O operations is limited by the resource channel, but it is not a one-to-one relationship. Each stripe often cannot be placed on its own chipid. With FICON we do loadsharing, and with FCP we can use a round-robin method. But depending on the number of channels and physical disks, we may reuse the same FCP channel several times.

When striping, contention can also occur at the control unit or at the disk unit. Ensure that the disks are distributed to different control units and define the minidisks on separate disk units.

Figure 10-7 illustrates how LVM can increase I/O performance.

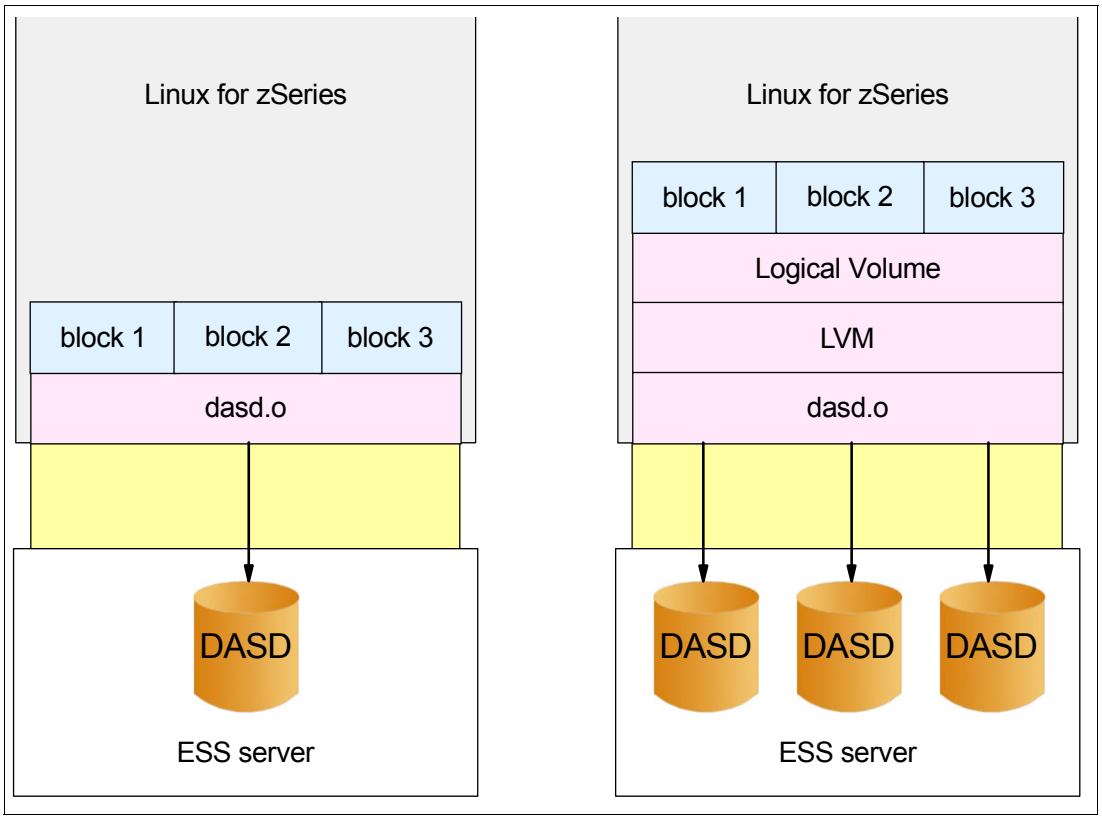


Figure 10-7 Using LVM for parallel DASD access

Using LVM, data is stripped across multiple physical DASD devices.

LVM2 versus EVMS

LVM2 is the most common Logical Volume Manager with Linux kernel 2.6. Alternatively, we can use the Enterprise Volume Management System (EVMS). We recommend using LVM2 and EVMS with striping.

- ESCON

  Use more or at least as many stripes as ESCON channels are available.

- FICON and FCP

  Use a number of stripes equal to the number of disks. We recommend that you use LVM2 and not EVMS for multipathing, as an LVM2 setup uses fewer CPU resources.

- Stripe size

  Good stripe sizes are 32 kb and 64 kb.

Multipathing - failover mode versus multibus mode with SCSI disks

Multipath I/O is a method to connect a computer system and the mass storage device using more than one path through buses, controllers, switches, and other connection devices. Multipathing increases system stability and resilience. There are two options with multipathing available:

- Multibus

  Multibus is a load-sharing option. Be aware that multibus is not implementing load sharing, as known from System z FICON channel path groups.

- Failover

  Failover is a high-availability option.

The failover mode shows better performance, causes lower CPU costs, and ensures better throughput rates than multibus.

More information about how to use multipathing and measurement results can be found here:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_how_dasd_multipath.html

10.4.8 Parallel Access Volume (PAV)

The concurrent operation's capabilities of the IBM TotalStorage® Enterprise Disk Storage Systems and IBM TotalStorage Enterprise Storage Server allow the system to access volumes in parallel, which means that it can concurrently transfer data to or from the same volume from the same system or system image. A volume that can be accessed parallelly is called Parallel Access Volume.

When using PAV, the application or system can support simultaneous access to the logical volume by multiple users or processes. Read operations can be accessed simultaneously, but write operation can be accessed parallelly only to different domains. Writes to the same domain still have to be serialized in order to maintain data integrity. The domain of an I/O operation is the specified extents to which the I/O operation applies.

With PTF for APAR VM63952, z/VM 5.2 supports PAV as minidisks for guest operating systems such as Linux and z/OS, which can exploit the PAV architecture. To use the PAV feature prior to z/VM 5.2, the devices must be dedicated to the guests.

IBM DASD PAV volumes must be defined to z/VM as DASDs with 3390 Model 2, 3, or 9 on a Storage Controller of 3390 Model 3 or 6, 2105, 2107, or 1750. It also supports 3380 track-compatibility mode for the 3390 Model 2 or 3 DASD.

With the PAV feature, storage systems can present the same physical disk volume as a base device and one or more alias devices. On System z, the base device and the aliases all can be assigned separate device numbers. Figure 10-8 illustrates the relationship between the base and alias devices for a volume.

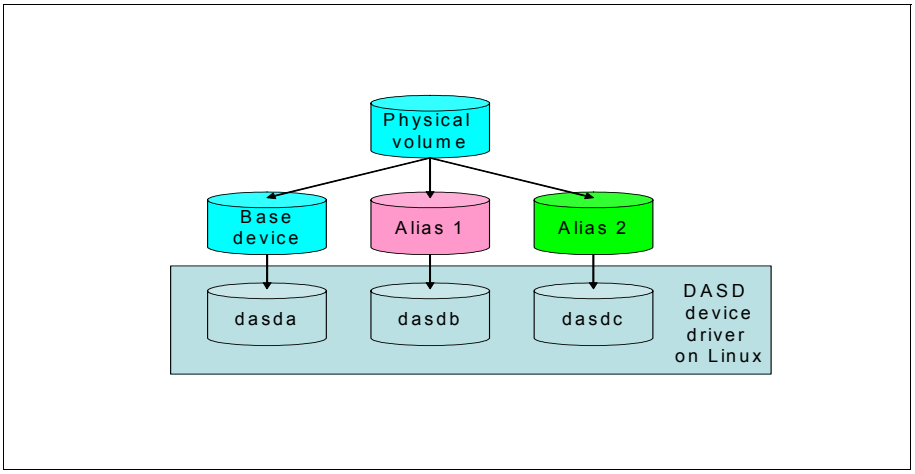


Figure 10-8 Relationship between the base and alias devices for a physical volume

In Figure 10-8, a disk volume can be accessed as a base device or one of two aliases. The base represents the real DASD volume space, and the aliases are *shadows* of the base. The aliases have no owned data space, but are mapped to the same physical volume as accessed by the base.

If a Linux system has access to a base device and its aliases, the DASD device driver on it senses the base device and each alias as a different and independent DASD, and assigns a different device name to each of them, as shown on Figure 10-8. After a DASD is partitioned and all the base devices and aliases are newly initialized, the DASD device driver senses the partitions and assigns additional device names for each of the partitions on both the base device and aliases, as illustrated on Figure 10-9.

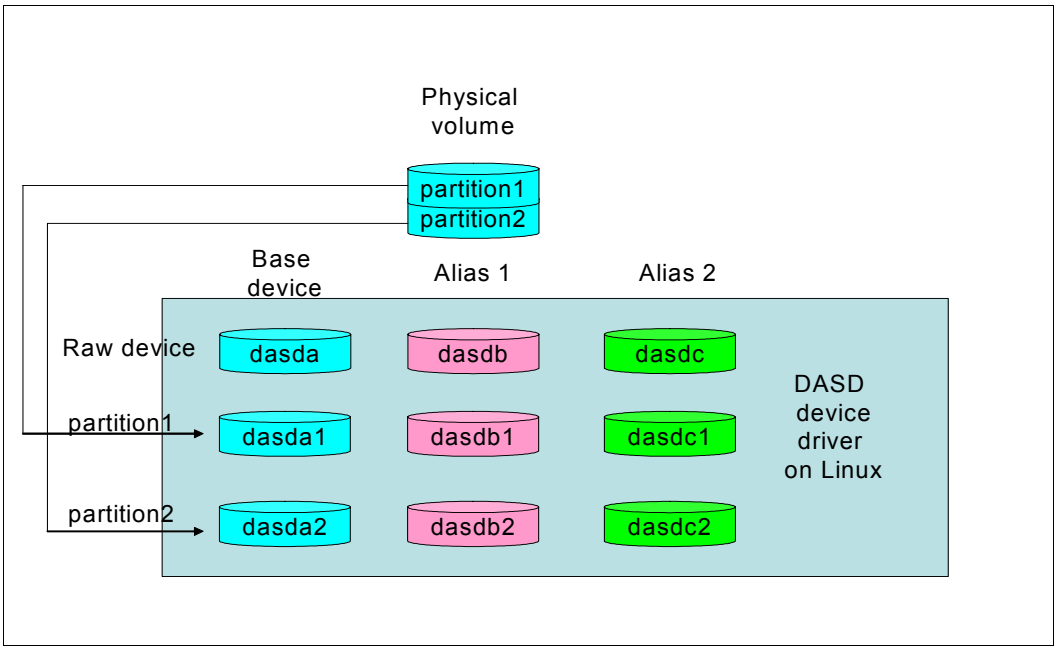


Figure 10-9 Relationship between the base and alias devices after partitioned seen from Linux guest

How to use PAV

PAV volumes can be used as both dedicated DASDs and minidisks. The difference is who should be responsible for optimizing the I/O performance. When using PAV volumes as dedicated DASDs, the performance benefits of PAV depend on how the Linux guests manage them, while in the case of using PAV volumes as minidisks, z/VM should take the responsibility of optimizing the I/O performance for the volumes used by multiple guests.

Use PAV volume as dedicated DASD

PAV devices can be used as dedicated DASD to the Linux guests that are running on z/VM. Once the base and associated aliases are attached to the Linux guest, it should manage their use. z/VM does not optimize the I/O flowing through the base and alias subchannels. A real volume cannot be dedicated to multiple guests. In such a dedicated environment, the performance benefits of PAV entirely depend on the Linux guest.

Example 10-3 and Figure 10-10 on page 161 illustrate a Linux guest with two dedicated physical volumes, each of which has a base device and two alias devices. z/VM dedicates the two real volumes as six virtual devices to the Linux guest, on which the DASD device driver senses them as different and independent DASDs.

Example 10-3 An entry for Linux guest in USER DIRECTORY

```
USER LINUX01 LNX4ME 2G 2G G
  INCLUDE LNXDFLT
  DEDICATE 1000 E100
  DEDICATE 1001 E200
  DEDICATE 1002 E201
  DEDICATE 1003 E101
  DEDICATE 1004 E202
  DEDICATE 1005 E203
```

Example 10-3 on page 160 shows an entry for the Linux guest in the USER DIRECTORY of z/VM. In this example, physical volumes E100 and E101 are dedicated to Linux guest LINUX01. Each of the physical volumes has two aliases, from E200 to E203, and are also dedicated to the Linux guest, as shown in Figure 10-10. z/VM dedicates these six devices as virtual devices from 1000 to 1005, which are seen from Linux as six different volumes. These volumes can be configured for multipathing, and then organized into a volume group for striping by using LVM or other logical volume manager. The Linux guest should manage and optimize these PAV volumes to get better performance by itself.

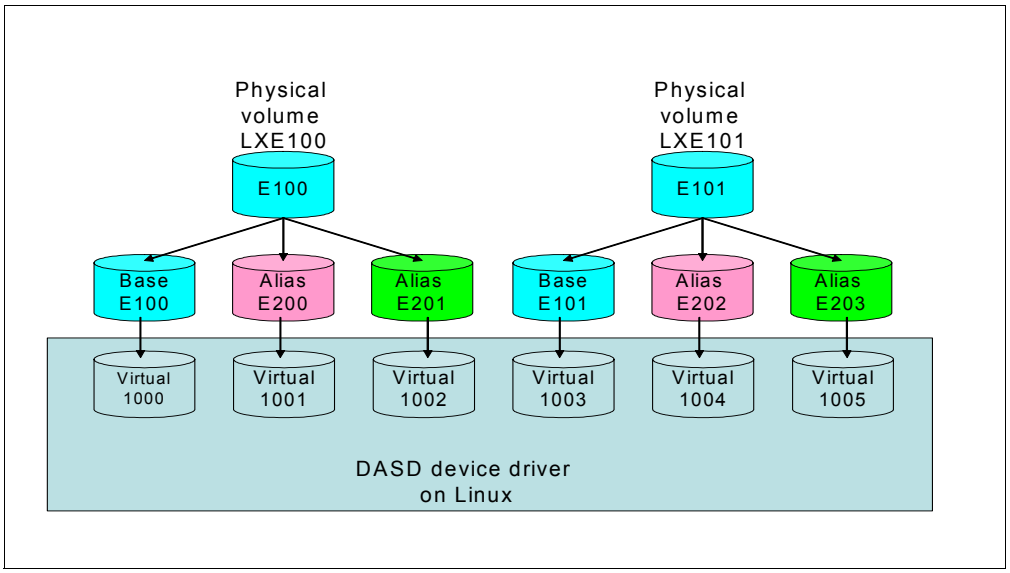


Figure 10-10 The relationship between physical volumes and virtual volumes seen from a Linux guest

Use PAV volumes as minidisks

When PAV volumes are used as minidisks, they support both full-pack and non-full-pack minidisks in the same manner. A guest virtual machine can define one or more minidisks on the same physical volumes. The physical PAV volumes have a real base device and one or more real aliases. Each minidisk on the physical volume has one virtual base device and zero or more virtual aliases. The number of virtual aliases for a guest should not exceed the number of the real aliases defined in the hardware for the real volume.

All I/O operations on minidisks through virtual base devices or virtual aliases are optimized by z/VM. z/VM automatically selects an appropriate subchannel from the real base device or real aliases. I/O performance can be gained only when full-pack minidisks are shared among guests by using LINK statements or when multiple non-full-pack minidisks are on a real PAV volume. Figure 10-11 on page 162 illustrates using PAV volumes as minidisks. There are two physical volumes, VOL001 and VOL002, each of which has a real base device and two real aliases. In the USER DIRECTORY of z/VM, user DEV1 is defined with a full-pack minidisk 100 on volume VOL001, which is linked by three other guests. These three guests define a 100-cylinder minidisk on volume VOL002 separately. By sharing minidisks on a real PAV volume in these two ways, I/O performance can be enhanced.

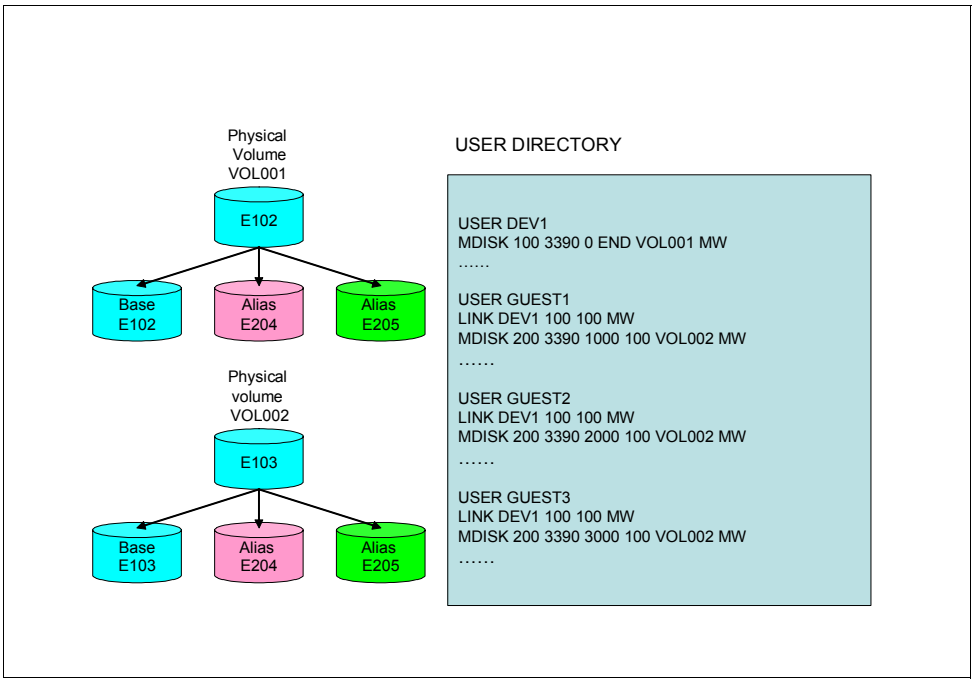


Figure 10-11 Using PAV volumes as minidisks

Linux can be one of the guests that uses PAV volumes as minidisks and depends on z/VM for I/O performance optimizing. Note that for other operating system guests, including z/VM v4.4, v5.1, v5.2 without PAV support, VSE, TPF, and CMS, they should rely on the underlying z/VM for PAV performance. z/OS guests can be configured to exploit PAV as well as Linux.

Performance of PAV

Generally speaking, disk performance depends on the time required to do a single I/O operation on a single disk volume in z/VM. This time is generally understood as *response time*, which is composed of *queue time* and *service time*. Queue time is the time a guest's I/O spends waiting to access the real volume. Service time is the time required to finish the real I/O operation. PAV can help to increase the access pathes to a real volume by using the base device and multiple alias devices, which leads to reducing the queue time. But if a volume is not experiencing queuing, adding aliases does not help to enhance the volume's performance, for the time required to perform an I/O operation is not appreciably changed by the queue time. In some cases, increasing the number of aliases for a volume might result in the increased service time for that volume. Most of the time, the decrease in wait time outweighs the increase in service time, so the response time improves.

The z/VM Performance Toolkit provides a report on the response time and related performance statistics for the real devices. Use the command DEVICE in the performance toolkit to access the report.

Example 10-4 DEVICE report from performance toolkit

```
FCX108      CPU 2094  SER 2991E  Interval 16:19:45 - 16:20:45    Perf. Monitor
____ .                    .        .     .    .    .    .    .    .    .    .
<-- Device Descr. -->  Mdisk Pa- <-Rate/s-> <------- Time (msec) -------> Req.
Addr Type   Label/ID   Links ths  I/O Avoid Pend Disc Conn Serv Resp CUWt Qued
C703 3390   LXC703         1   4 60.1  36.3   .4  5.7  2.3  8.4  8.4   .0   .0
C704 3390   LXC704         1   4  1.2    .0   .3  3.0  1.5  4.8  4.8   .0   .0
C705 3390   LXC705         1   4   .5    .0   .3  1.4   .7  2.4  2.4   .0   .0
C706 3390   LXC706         1   4   .4    .0   .3  1.7   .6  2.6  2.6   .0   .0
```

Example 10-4 illustrates the report from the performance toolkit. Among the columns, the following are interesting when tuning the PAV for disk performance:

- *I/O* is the I/O rate for the device, in operations per second.
- *Serv* (service time) is the average amount of time (milliseconds) the device uses when performing a single I/O.
- *Resp* (response time) is the average amount of time (in milliseconds) that the guest machine perceives is required to perform an I/O to its minidisk on the volume. Response time includes service time plus wait time (also known as queue time).
- *Req.Qued* is the average length of the wait queue for the real device. This is the average number of I/Os waiting in line to use the volume.

Generally speaking, when you notice that a volume is experiencing queuing by observing the Req.Qued from the performance toolkit, add aliases to that volume until you run out of alias capability, or until the queue is empty. The strategy for tuning PAV is to estimate the queue depth for the number of aliases, and add aliases for the volume until the volume's I/O rate maximizes or the volume's wait queue is empty, whichever comes first.

For more information see:

- z/VM PAV support

  http://www.vm.ibm.com/storman/pav/pav2.html
- Performance report

  http://www.vm.ibm.com/perf/reports/zvm/html/index.html

10.4.9 Linux I/O schedulers

The I/O scheduler optimizes disk access. Issuing each request directly to the disk results in a bad throughput, because this strategy causes a high amount of small requests, where in the worst case each request demands a completely different disk area. Today data transfer rates from physical disks are very high, so transfer times are very short. The dominating part of the access time is the disk latency, mainly caused by disk head movements of about 8 ms on average. The strategy for optimization is to minimize the number of I/O operations and disk head movements.

I/O schedulers mostly implement two functions:

- Request merging

  This merges two adjacent requests into one request.

- Elevator

  This orders the requests to minimize seek times, considering the physical data location. (Seek time is the duration of a head movement to reach an addressed sector.) A *prevent starvation* functionality is required, which ensures that requests that are not in the *seek reducing* order are served in an adequate time frame. Here the various I/O schedulers have different strategies.

New schedulers since kernel 2.6

There are new schedulers available with Linux from kernel 2.6:

- Noop scheduler

  The noop scheduler implements only the request merging function.

- Deadline scheduler

  The deadline scheduler implements request merging and the elevator, and in addition tries to prevent request starvation (which can be caused by the elevator algorithm) by introducing a deadline for each request. In addition, it prefers readers. Linux does not delay write processes until their data are really written to the disk, because it caches the data, while readers have to wait until their data are available.

- Anticipatory scheduler (as scheduler)

  The anticipatory scheduler implements request merging and the elevator, and in addition optimizes for physical disks by avoiding too many head movements. It tries to solve situations where many write requests are interrupted by a few read requests. After a read operation it waits for a certain time frame for another read and does not switch back immediately to write requests. This scheduler is not intended to be used for storage servers.

- Complete fair queuing scheduler (cfq scheduler)

  The complete fair queuing scheduler implements request merging and the elevator, and in addition embarks on the strategy to allow all users of a particular drive about the same number of I/O requests over a given time.

Default

The default in kernel 2.6 was for a long time the anticipatory scheduler. A Novell/SUSE SLES9 and Red Hat RHEL4 installations have overwritten this with the cfq scheduler. With kernel 2.6.18 and later the default is the cfq scheduler.

Check for the current scheduler

Searching (grep) in the /var/log/boot.msg file for the phrase *io scheduler* finds a line like:

```
<4>Using deadline io scheduler
```

That returns the name of the scheduler in use.

How to select the scheduler

The I/O scheduler is selected with the boot parameter elevator in zipl.conf.

Example 10-5 /etc/zipl/conf with elevator parameter

```
...
[ipl]
    target = /boot/zipl
    image = /boot/image
    ramdisk = /boot/initrd
    parameters = "maxcpus=8 dasd=5000 root=/dev/dasda1 elevator=deadline"
```

Where elevator::= as | deadline | cfq | noop.

For more details see /usr/src/linux/Documentation/kernel-parameters.txt.

10.4.10 Linux disk I/O setup

The throughput and the cost for disk I/O depend on whether we use direct I/O to the disk or whether we use a file system. The prerequisite to use direct I/O is that the application is ready to take over some of the functions a file system usually offers. Experiments using DBMS showed promising results for throughput and costs bypassing the file system for both DASD and SCSI disks.

10.4.11 Linux file systems

Beside the traditional ext2 file system we have today, various journaling file systems are included in the Linux distribution. Since they all have pros and cons, we cannot recommend a special file system. The performance depends on many parameters. At most, it depends on the character of the files, the workloads, and the kind of access (sequential, random, mixed). Journaling file systems have an overhead compared to non-journal file systems. But its use is strongly recommended since data loss occurs less probably and the recovery time is much shorter in the case of an error.

The most used is today the ext3 files system because of stability and because it has been developed on top of ext2. Many factors, like applications and tools, influence the decision of which file system to use. Performance varies, so it may be worthwhile to try different file systems with important and time-critical workload.

Another nice feature ext3 offers is the seamless migration from ext2. Just issue a “tune2fs -j <partition>”. A journal is created and the file system is mountable as ext3. If you have hundreds of gigabytes of data existing in ext2 and you do not want to interrupt production, back up, reformat, and restore the data, and ext3 is available for you.

Results that show the performance influence on different file systems are available at:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_res_journaling.html#results

10.4.12 Read-ahead setup

Several components of the environment are candidates for performing read aheads. This makes sense if data is required to read in sequential order or if the data entity is big (huge database entries):

- Applications
- LVM
- Block device layer
- ESS

On the other hand, the performance of the low hit workloads with high I/O load suffers from read-ahead operations.

Each of mentioned components issues read-ahead activities. This leads to more I/O activity than requested. All components assume that subsequent data units will be needed soon:

- With each database read access the application may try to fill its cache with pages for a set of data.
- For each page, the logical volume manager fetches a set of pages for subsequent file blocks. The default of 1024 pages is much to high and results in a performance degradation for this workload (but, once changed, this could not be set higher than 255 pages).
- The Linux block device layer may also add some pages to each read request. For each file block the ESS gets a set of subsequent disk blocks (several disks if the logical volume is striped).

The overhead for managing the caches and the thrashing of the caches is high with data that is most likely not needed. The big amount of physical disk read requests together with the physical limitations in terms of seek and latency times limited the speed of the database accesses in our test. Lowering the default settings of read ahead allows you to keep the data transfer path free for the requested data.

11

Network considerations

In this chapter we provide information about networking options available on Linux for System z and z/VM based architectures. These options can influence overall performance and should be identified and defined before implementing a solution. In this chapter, the following topics are discussed:

- Guidelines for network selection
- Network configuration parameters
- z/VM 5.3 enhancements
- Hipersockets
- Tuning recommendation

© Copyright IBM Corp. 2003, 2008. All rights reserved.

11.1 Selecting network options

This section describes the different options that you can use to attach Linux on System z to an external network. Linux can use the network directly or by services managed by z/VM. The choice between the different solutions can influence overall performance.

11.1.1 Physical networking

Linux can be installed on LPARs and under z/VM. With Linux on LPAR the only choice we have is the direct attach to the network connecting the operating system directly to the OSA adapter, as shown in Figure 11-1.

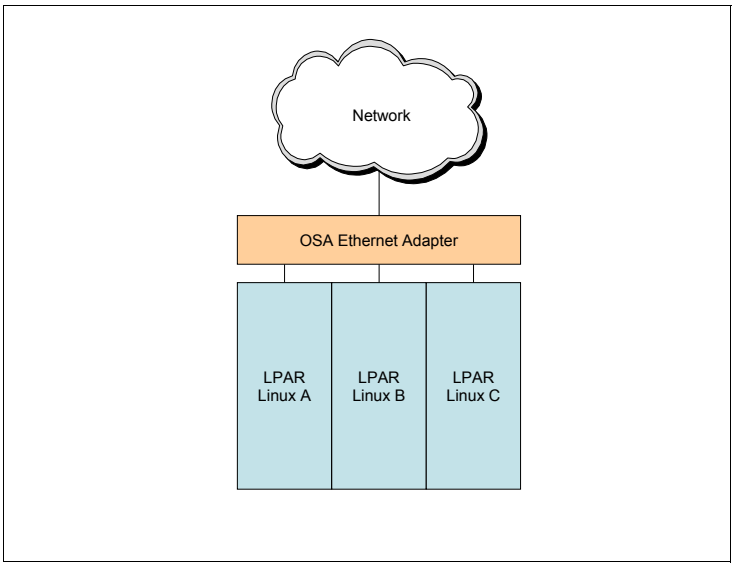


Figure 11-1 Direct attach to network

Direct-attached networks are possible with z/VM, also. The DEDICATE command on the USER DIRECT definition for the Linux guest makes this possible. This means that Linux directly controls the network interface. This can be used when you do not have many images to be managed.

Using only one adapter may result in a single point of failure. If an OSA adapter fails, Linux will no longer be available. To avoid this, it is possible to duplicate the network connection and use a high availability solution such as a virtual IP address. This solution normally introduces protocols to manage the dynamic routing, and the result is complete high availability with some overhead introduced by products used to mange the routing, like zebra and the ospf daemon. If you want to use a high availability solution on Linux under z/VM we suggest using a virtualized network managed by VSWITCH, as described in 11.1.2, “Virtual networking” on page 171.

You need to consider that a Direct OSA connection is a little bit faster than VSWITCH. Figure 11-2 is a graph that shows transaction request per second managed by Direct OSA connection versus VSWITCH. This performance data is made using a TCP request and response test. For more information about the test used in this chapter, refer to 11.6, “Workloads and performance tools” on page 189.

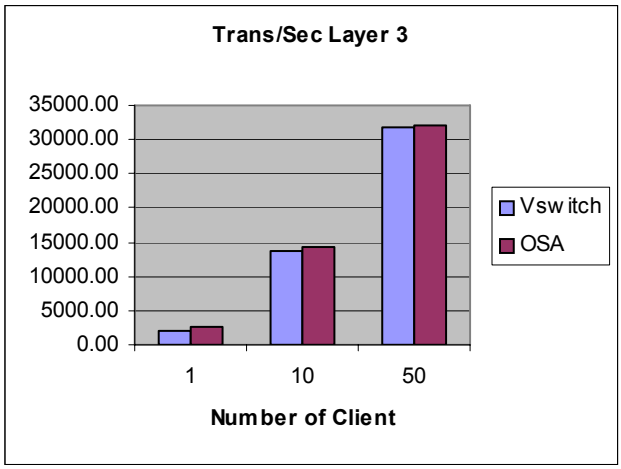


Figure 11-2 Trans/sec report from RR test

Figure 11-3 shows the CPU usage in the same scenario as in Figure 11-2. As you can see, VSWITCH connections consume more CPU than direct connections.

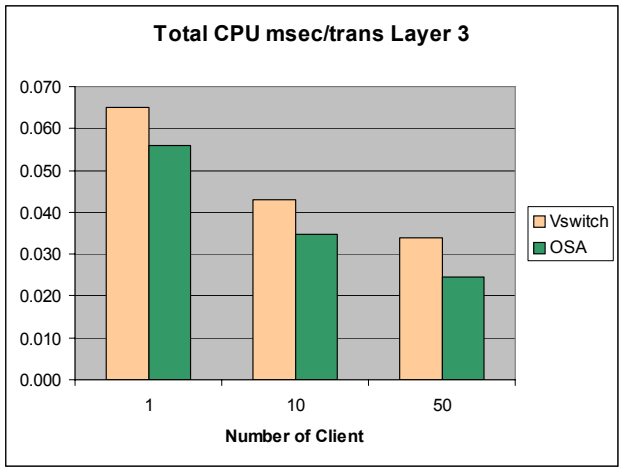


Figure 11-3 Total CPU msec report from RR test

In a streaming benchmark tests, the differences are more evident. The same two graphs are provided using the streaming benchmark tests in Figure 11-4 and Figure 11-5.

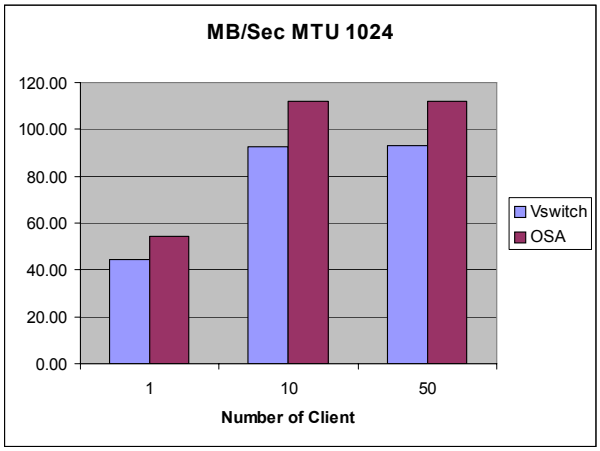


Figure 11-4 MBps report from STR test

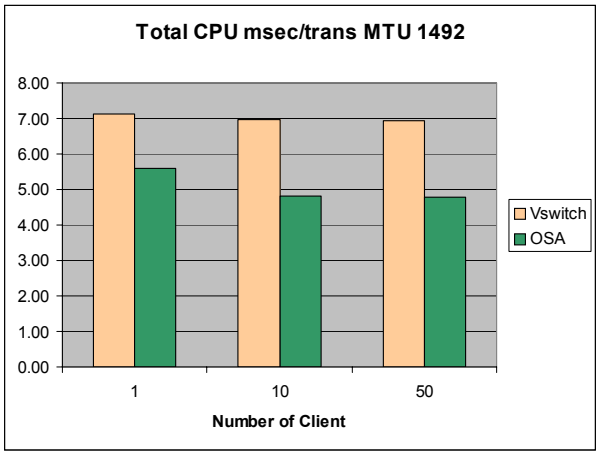


Figure 11-5 Total CPU msec/trans report from STR test

In Figure 11-6, all network connections were measured with connect-request-response (CRR) tests.

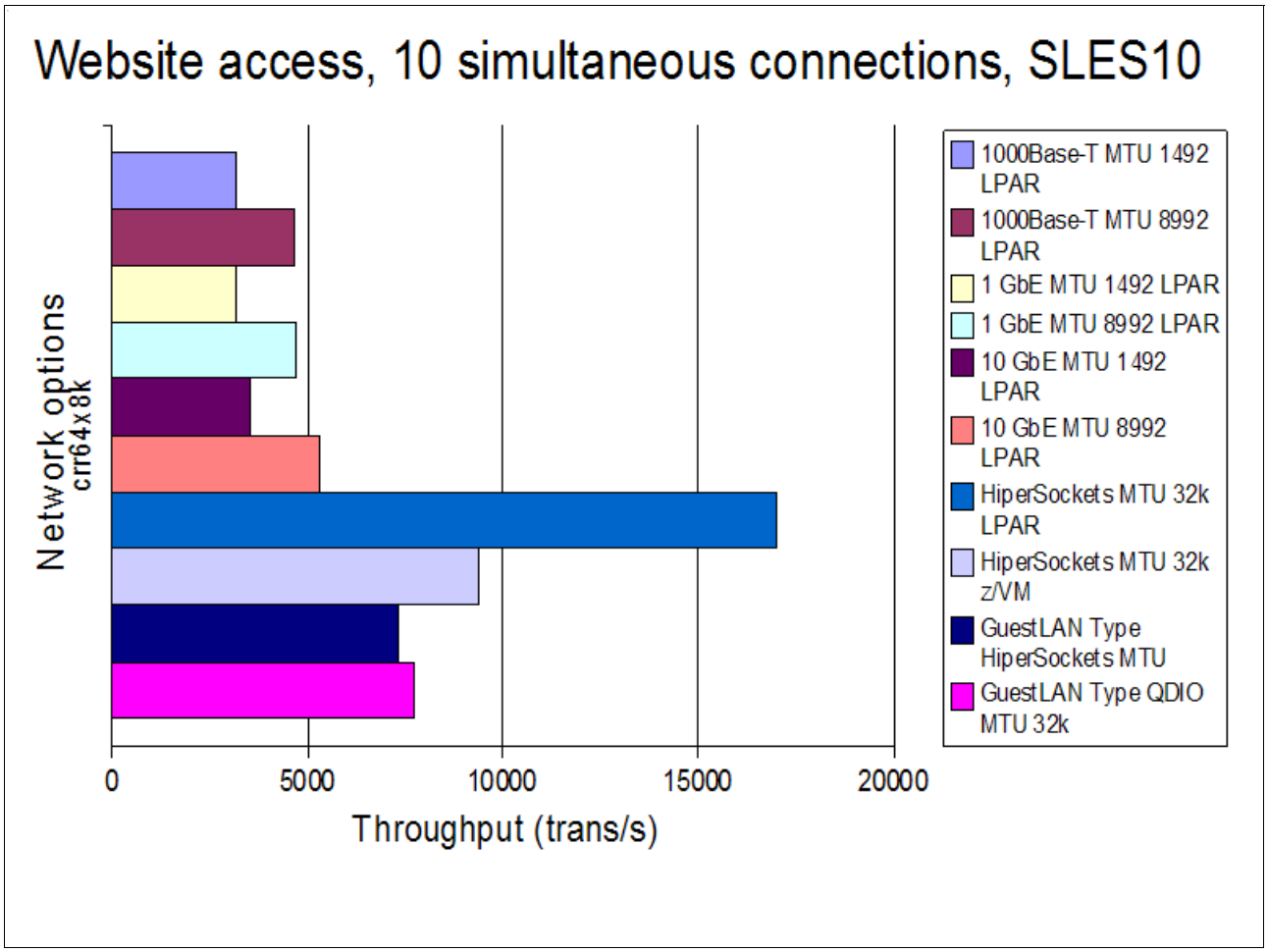


Figure 11-6 OSA connections measurements

As reported in 11.6, “Workloads and performance tools” on page 189, CRR is a particular benchmark test where Web connections are simulated. This test give you the perceptions of what happens in a real application scenario. Figure 11-6 shows that Hipersockets are the best connections in terms of throughput. In these results, you can see also measurements of the new 10 Gigabit OSA card that performs better than other OSA cards.

11.1.2 Virtual networking

Virtual networking is available when z/VM is used to control the accesses of Linux in the network. Many improvements were deployed in the past few years to consolidate the network architecture and make it more efficient. From CTC and IUCV point-to-point link we can use new enhancements based on the Virtual Switch. The Virtual Switch (VSWITCH) was introduced to eliminate Linux images used to route the inside network to the outside network or z/VM TCP/IP routing functions. With this architecture you do not need to use any other images to perform routing processes because the VSWITCH acts as a real Network Switch built in z/VM. Linux routers introduce some overhead, especially in a redundant scenario where two or more Linux routers should be used to ensure high availability.

Figure 11-7 shows the possibilities to attach Linux images to the network.

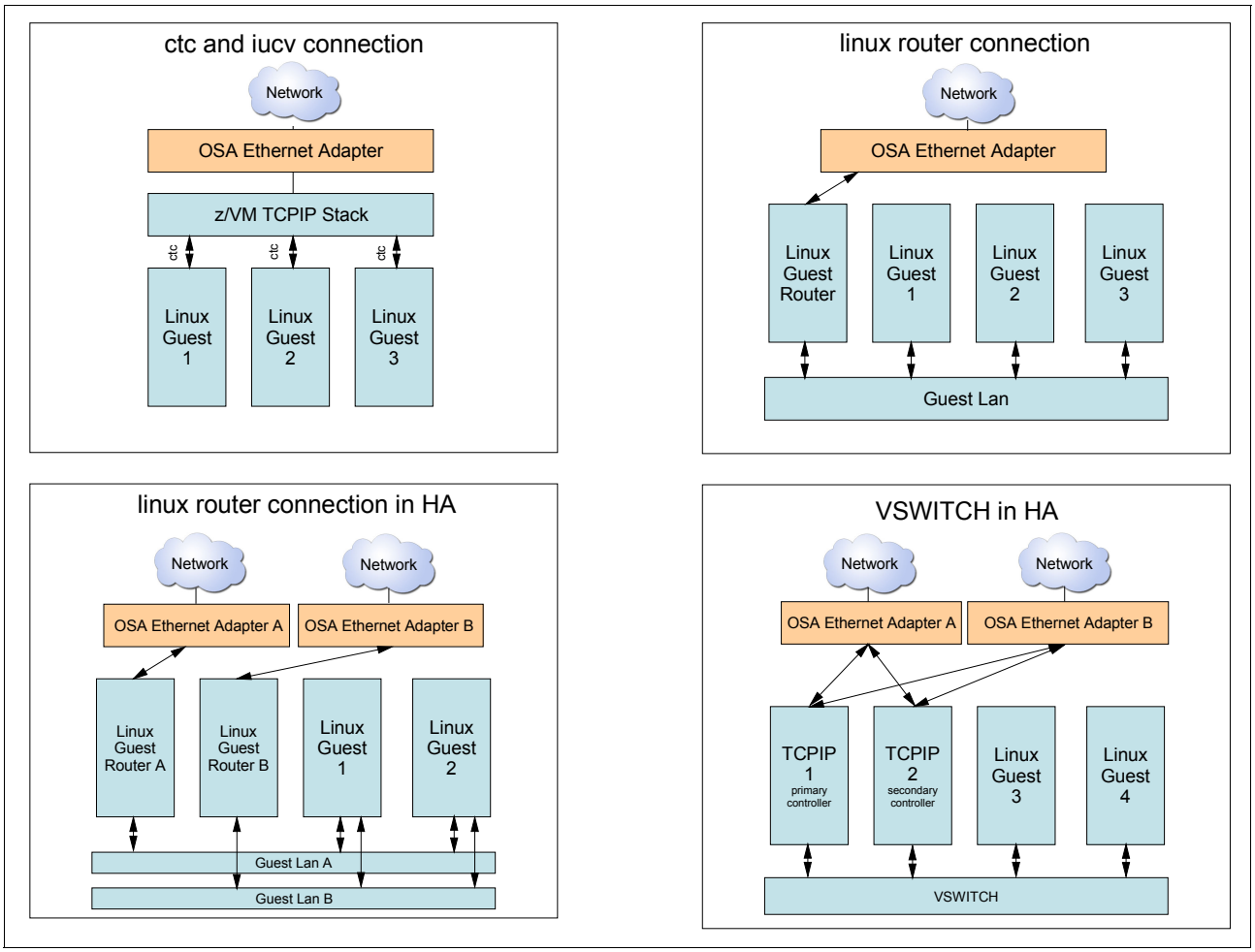


Figure 11-7 Virtual networking scenarios

In the first example, the TCP/IP virtual machine is responsible for routing traffic between the Linux images and the rest of the network. With this configuration, each Linux uses a point-to-point link (ctc or iucv) to TCP/IP Virtual Machine. At the time of writing this book, this configuration is no longer used, but is still supported. We suggest migrating, due to performance reasons, to other virtualized solutions, as described below.

If you do not want to use the TCP/IP stack for routing, you can use Linux. In a server farm based on Linux and z/VM, it can be a good solution to use Linux as a router to external network and link (simply using the default gateway statement) the other Linux images to it. All Linux images can use the OSA driver connected to a Guest LAN managed by z/VM. If you want use a high-availability scenario, you need to:

1. Duplicate the guest LAN where Linux is attached.
2. Duplicate the Linux router.
3. Use one VIPA address for each Linux.
4. Implement a dynamic routing protocol, for example, ospf.

This solution offers a perfect high availability scenario but introduces some features that can influence network and CPU performance. First of all, introducing another Linux means introducing other workload to be managed. Ospf and all the daemons associated (zebra, for

example) continuously poll the network to see whether something changed. This operation wakes up the processor many times during the normal operations.

To avoid this, the term VSWITCH was introduced starting from z/VM V4.4. The term indicates a Virtual Switch (working as a hardware network switch) able to bridge a local network (guest LAN) to an external network. This solution offers a built-in redundancy because the VSWITCH can be configured to work with a primary OSA interface and in case of failure with a secondary OSA interface. Major enhancements are provided with z/VM 5.3 with the Virtual Switch Link Aggregation support, as described in 11.3, "z/VM Virtual Switch link aggregation" on page 182.

Table 11-1 lists the advantages and disadvantages of using the scenario described.

Table 11-1 Performance comparison

Network types	Availability	Performance
Point-to-point with ctc drivers	Single point of failure on z/VM TCP/IP stack	Poor
Point-to-point with iucv drivers	Single point of failure on z/VM TCP/IP stack	Poor but better than ctc
Linux router	Single point of failure on Linux router	Faster than point-to-point
Redundant Linux routers	No single point of failure	Poor (depending on type and number of CPUs)
Virtual Switch (VSWITCH)	Single point of failure on TCP/IP controller	Good
Redundant Virtual Switch	No single point of failure	Good
Virtual Switch Link aggregation[a]	No single point of failure	Very good (increase depends on number of OSA cards)

a. More information is discussed in 11.3, "z/VM Virtual Switch link aggregation" on page 182.

11.2 Network configuration parameters

In this section not all of the network driver is reviewed, only the most commonly used. The purpose of this section is to analyze which parameter can be used for a correct configuration and optimization.

11.2.1 Qeth device driver for OSA-Express (QDIO)

Queued Direct I/O is a highly efficient data transfer mechanism that satisfies the increasing volume of TCP/IP applications and increasing bandwidth demands. It reduces system overhead and improves throughput. The components that make up QDIO are:

- Direct Memory Access
- Priority queuing
- Dynamic OSA address table update
- LPAR-to-LPAR communication
- IP assist functions

The qeth network device driver supports System z OSA-Express and Osa-Express2 features in QDIO mode and Hipersockets. This module is the most commonly used for network connection in the Linux for System z environment, both in the physical and the virtual networking configuration.

The qeth device driver requires three I/O subchannels for each CHPID in QDIO mode. One subchannel is for control reads, one for control writes, and the third for data. This device uses the QDIO protocol to communicate with the adapter (OSA or Hipersockets).

Figure 11-8 shows the I/O subchannel interface.

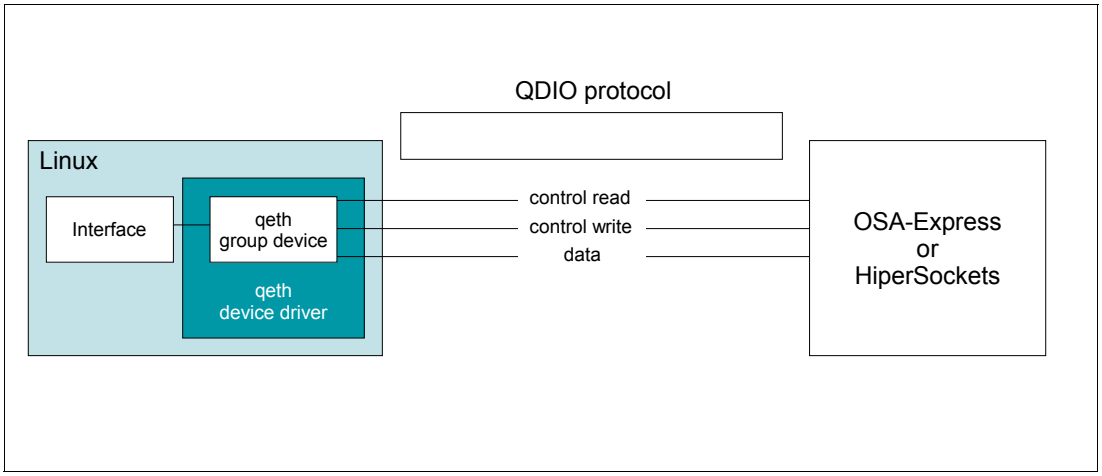


Figure 11-8 I/O subchannel interface

With the new sysfs file system all information about drivers is collected on /sys/bus/ccwgroup/drivers/qeth.

You can interact with this file system to modify, add, or query drivers parameters. Example 11-1 shows the file system structure related to our channel 0.0.c200 used in our environment.

Example 11-1 List command to show sysfs files related to driver

```
# ls -la /sys/bus/ccwgroup/drivers/qeth/0.0.c200/
total 0
drwxr-xr-x 6 root root    0 Sep 28 07:59 .
drwxr-xr-x 6 root root    0 Sep 28 08:01 ..
-rw-r--r-- 1 root root 4096 Sep 28 07:59 add_hhlen
drwxr-xr-x 2 root root    0 Sep 28 07:59 blkt
-rw-r--r-- 1 root root 4096 Sep 28 07:59 broadcast_mode
-rw-r--r-- 1 root root 4096 Sep 28 07:59 buffer_count
lrwxrwxrwx 1 root root    0 Sep 28 07:59 bus -> ../../../bus/ccwgroup
-rw-r--r-- 1 root root 4096 Sep 28 07:59 canonical_macaddr
-r--r--r-- 1 root root 4096 Sep 28 07:59 card_type
lrwxrwxrwx 1 root root    0 Sep 28 07:59 cdev0 ->
../../../devices/css0/0.0.0009/0.0.c200
lrwxrwxrwx 1 root root    0 Sep 28 07:59 cdev1 ->
../../../devices/css0/0.0.000a/0.0.c201
lrwxrwxrwx 1 root root    0 Sep 28 07:59 cdev2 ->
../../../devices/css0/0.0.000b/0.0.c202
-rw-r--r-- 1 root root 4096 Sep 28 07:59 checksumming
-r--r--r-- 1 root root 4096 Sep 28 07:59 chpid
```

```
lrwxrwxrwx 1 root root    0 Sep 28 07:59 driver ->
../../../bus/ccwgroup/drivers/qeth
-rw-r--r-- 1 root root 4096 Sep 28 07:59 fake_broadcast
-rw-r--r-- 1 root root 4096 Sep 28 07:59 fake_ll
-r--r--r-- 1 root root 4096 Sep 28 07:59 if_name
drwxr-xr-x 2 root root    0 Sep 28 07:59 ipa_takeover
-rw-r--r-- 1 root root 4096 Sep 28 07:59 large_send
-rw-r--r-- 1 root root 4096 Sep 28 07:59 layer2
lrwxrwxrwx 1 root root    0 Sep 28 07:59 net:eth0 -> ../../../class/net/eth0
-rw-r--r-- 1 root root 4096 Sep 28 07:59 online
-rw-r--r-- 1 root root 4096 Sep 28 07:59 performance_stats
-rw-r--r-- 1 root root 4096 Sep 28 07:59 portname
-rw-r--r-- 1 root root 4096 Sep 28 07:59 portno
-rw-r--r-- 1 root root 4096 Sep 28 07:59 priority_queueing
--w------- 1 root root 4096 Sep 28 07:59 recover
-rw-r--r-- 1 root root 4096 Sep 28 07:59 route4
-rw-r--r-- 1 root root 4096 Sep 28 07:59 route6
drwxr-xr-x 2 root root    0 Sep 28 07:59 rxip
-r--r--r-- 1 root root 4096 Sep 28 07:59 state
--w------- 1 root root 4096 Sep 28 07:59 uevent
--w------- 1 root root 4096 Sep 28 07:59 ungroup
drwxr-xr-x 2 root root    0 Sep 28 07:59 vipa
```

Table 11-2 shows all parameters available on the sysfs file system. This parameter can be dynamically updated with the sysfs file system. The parameters are divided in three different categories:

- Display: in order to get information about the OSA configuration
- Configuration: to configure other features on OSA
- Performance: to tune the OSA adapter

Table 11-2 qeth device parameter

File	Category	Description
add_hhlen	Performance	Reserve additional hardware-header space in front of every packet in socket buffer.
broadcast_mode	Configuration	For token ring broadcast.
buffer_count	Performance	You can assign from 8 to 128 buffers for inbound traffic.
canonical_macaddr	Configuration	For token ring networks.
card_type	Display	Display the card type used.
checksumming	Performance	Offload the checksum calculation to an OSA card.
chpid	Display	Display the channel path ID used for this device.
fake_broadcast	Configuration	Add broadcast attribute to non-broadcast network.
fake_ll	Configuration	Add dummy LLC header in all incoming packets.
if_name	Display	Display Linux interface name (for example, eth0).

File	Category	Description
ipa_takeover	Configuration	Directory contains vipa configurations.
large_send	Performance	Offload the TCP segmentation (available only for Osa-Express2 card).
layer2	Configuration	Enable layer2 including LLC header.
online	Configuration	Set online/offline device.
performance_stats	Configuration	Start and stop QETH performance statistics.
portname	Configuration	Display/modify OSA portname.
portno	Configuration	Specify the relative port number (available only for OSA-Express ATM).
priority_queueing	Performance	Give a different priority queue in main storage for outgoing IP packets.
recover	Configuration	Recover a device in case of failure.
route4	Configuration	Setup a Linux router IPV4.
route6	Configuration	Setup a Linux router IPV6.
rxip	Configuration	Configure a device for proxy ARP.
state	Display	Display status of network.

You can display all information about your OSA configuration by issuing the **lsqeth** command. Example shows output generated by a qdio driver used in a z/VM guest LAN scenario.

Example 11-2 lsqeth command output for guest LAN

```
# lsqeth
Device name                    : eth0
---------------------------------------------
        card_type              : GuestLAN QDIO
        cdev0                  : 0.0.c200
        cdev1                  : 0.0.c201
        cdev2                  : 0.0.c202
        chpid                  : 1C
        online                 : 1
        portname               : any
        portno                 : 0
        route4                 : no
        route6                 : no
        checksumming           : sw checksumming
        state                  : UP (LAN ONLINE)
        priority_queueing      : always queue 2
        fake_ll                : 0
        fake_broadcast         : 0
        buffer_count           : 16
        add_hhlen              : 0
        layer2                 : 0
        large_send             : no
```

Starting and stopping collection of QETH performance statistics

You can enable performance statistics for the qeth device using the **echo** command, as shown in Example , where we are assuming use of a c200 device.

Example 11-3 Enable performance statistics collection

```
# echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.c200/performance_stats
```

You can get statistic informations on the /proc file system, as shown in Example 11-4.

Example 11-4 Performance statistics output sample

```
# cat /proc/qeth_perf
For card with devnos 0.0.c200/0.0.c201/0.0.c202 (eth0):
  Skb's/buffers received                 : 4421/4230
  Skb's/buffers sent                     : 1228/1228

  Skb's/buffers sent without packing     : 1228/1228
  Skb's/buffers sent with packing        : 0/0

  Skbs sent in SG mode                   : 0
  Skb fragments sent in SG mode          : 0

  large_send tx (in Kbytes)              : 0
  large_send count                       : 0

  Packing state changes no pkg.->packing : 0/0
  Watermarks L/H                         : 2/5
  Current buffer usage (outbound q's)    : 0/0/0/0

  Inbound handler time (in us)           : 70223
  Inbound handler count                  : 4156
  Inbound do_QDIO time (in us)           : 1290
  Inbound do_QDIO count                  : 529

  Outbound handler time (in us)          : 2587
  Outbound handler count                 : 1228

  Outbound time (in us, incl QDIO)       : 37975
  Outbound count                         : 1228
  Outbound do_QDIO time (in us)          : 29530
  Outbound do_QDIO count                 : 1228
```

OSA parameter: add_hhlen

Some software makes use of free space in front of packets. For example, extra space can be beneficial for the Linux virtual server and IP tunneling. To reserve additional hardware-header space in front of every packet in the socket buffer set the add_hhlen attribute to an integer value up to 1024. The integer is the number of bytes to be added. Example 11-5 shows the command used for modify hardware-header space.

Example 11-5 Modify add_hhlen parameter

```
# echo <integer> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/add_hhlen
```

Attention: The device must be offline while you add additional hardware-header space.

OSA parameter: buffer_count

The qeth device driver assigns 16 buffers for inbound traffic to each qeth group device. Depending on the amount of available storage and amount of traffic, you can assign from 8 to 128 buffers.

To understand Linux memory consumption (in case you are planning to modify this parameter), consider that Linux memory usage for inbound data buffers for the devices is (number of buffers) x (buffer size).

The buffer size is equivalent to the frame size, which is:

- For an OSA-Express CHPID in QDIO mode: 64 KB
- For Hipersockets: depending on the Hipersockets CHPID definition, 16 KB, 24 KB, 40 KB, or 64 KB

Set the buffer_count attribute to the number of inbound buffers that you want to assign. Example 11-6 shows the command used to modify the buffer_count parameter.

Example 11-6 Modify buffer_count parameter

```
# echo <number> > /sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/buffer_count
```

Attention: The device must be offline while you specify the number of inbound buffers.

OSA parameter: checksumming

Checksumming can be performed via software (TCP/IP stack) or via hardware (if the OSA card support it). To check how the checksumming is performed, issue a command where device 0.0.c200 is assumed to be used, as shown in the Example 11-7.

Example 11-7 Display checksumming on device

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.c200/checksumming
sw checksumming
```

To change the checksumming parameter simply add the value for checksumming file with the **echo** command. Possible value are:

- hw_checksumming: Checksumming is performed by the OSA adapter.
- sw_checksumming: Checksumming is performed by the TCP/IP stack.
- no_checksumming: Checksumming is suppressed.

Attention: The device must be offline while you change the checksumming parameter.

Figure 11-9 shows the test performed in our scenario. We test software and hardware checksumming. As you can see, CPU is less used when checksumming is performed by the OSA adapter. In our case we use an OSA Express 1000BASE-T. The differences are more relevant if you use OSA Express 2 cards.

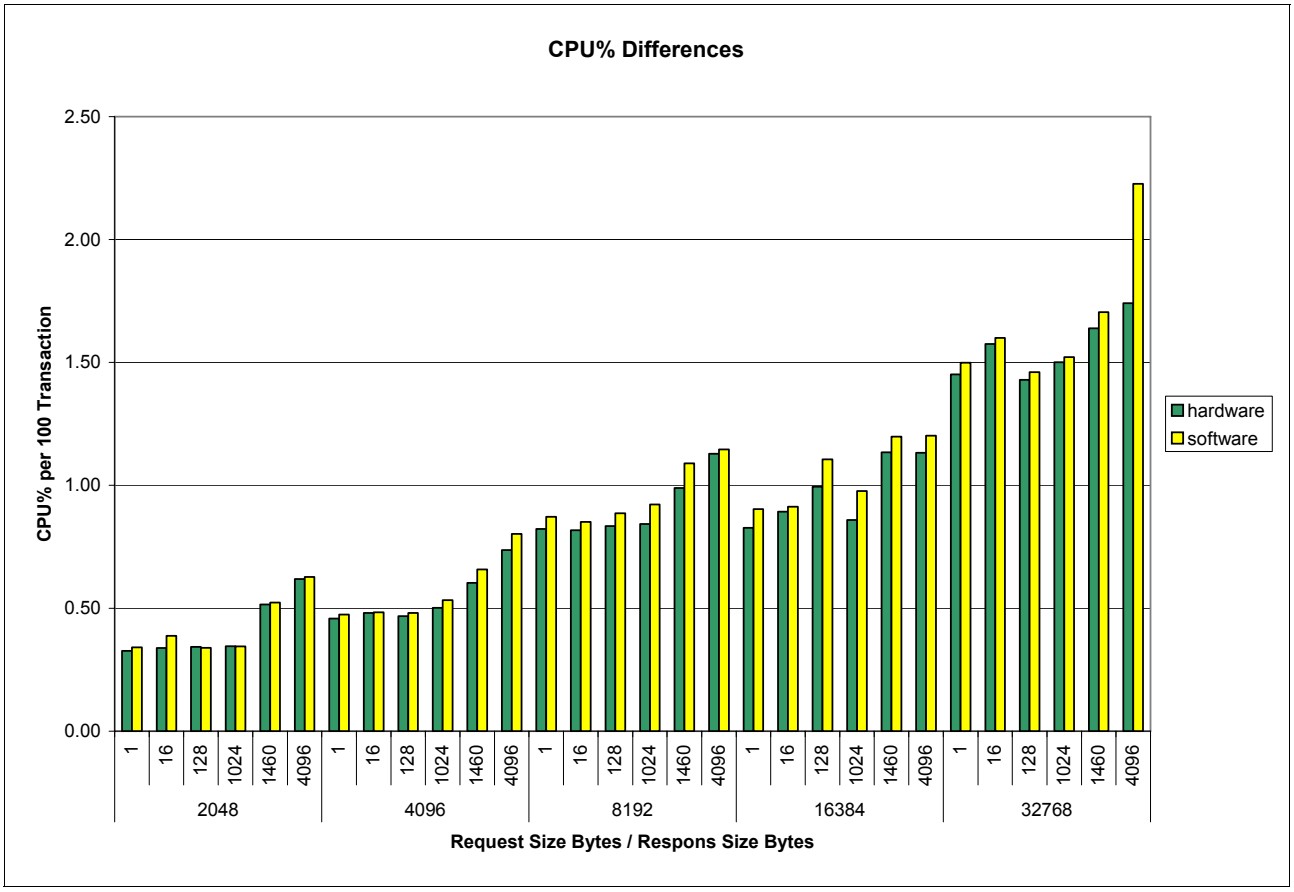


Figure 11-9 Hardware versus software checksumming

OSA parameter: large_send

As checksumming, you can offload the TCP segmentation from the Linux network stack to OSA-Express 2. A large send can increase performance for an interface with a high amount of large outgoing packets. The large_send parameter can be modified with the **echo** command assuming different values:

- no: No large send is provided. The Linux TCP/IP stack performs the segmentation.
- TSO: The OSA network adapter performs segmentation.
- EDDP: The qeth driver performs segmentation.

OSA parameter: priority_queuing

An OSA-Express CHPID in QDIO mode has four output queues in main storage. This parameter gives these queues different priorities. Queuing is relevant to high-traffic situations. You can check the priority_queuing with the **cat** command. Example 11-8 shows the priority queuing set in our 0.0.c200 device.

Example 11-8 Display priority queue related to 0.0.c200 device

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.c200/priority_queueing
always queue 2
```

This parameter can be modified with this value:

- prio_queuing_prec: to base the queue assignment on the two most significant bits of each packet's IP header precedent field.
- prio_queuing_tos: to select a queue according to the IP type of service assigned by some application to packets (auto managed queue).
- no_prio_queueing: the default. Queue 2 is used for all packets.
- no_prio_queueing:0: Queue 0 is used for all packets.
- no_prio_queueing:1: Queue 1 is used for all packets.
- no_prio_queueing:2: Queue 2 is used for all packets as the default.
- no_prio_queueing:3: Queue 3 is used for all packets.

Table 11-3 shows the relation between the queue and the service type. This is also useful to understand how the qeth device driver maps service types to the availability queues in the prio_queueing_tos parameter.

Table 11-3 IP service types and queue assignment for type of service queuing

Service type	Queue
Low latency	0
High throughput	1
High reliability	2
Not important	3

Attention: The device must be offline while you change the priority queue parameter. This function applies to OSA-Express CHPIDs in QDIO mode only.

11.2.2 LAN channel station (non-QDIO)

The LCS module is used when an OSA Adapter is configured as OSE. If you use LCS drivers, you can share the adapter between images, but you need to configure it via OSASF. With this device driver, the OAT table is not dynamically managed, and an opportune configuration should be taken. Figure 11-10 shows the I/O subchannel interface.

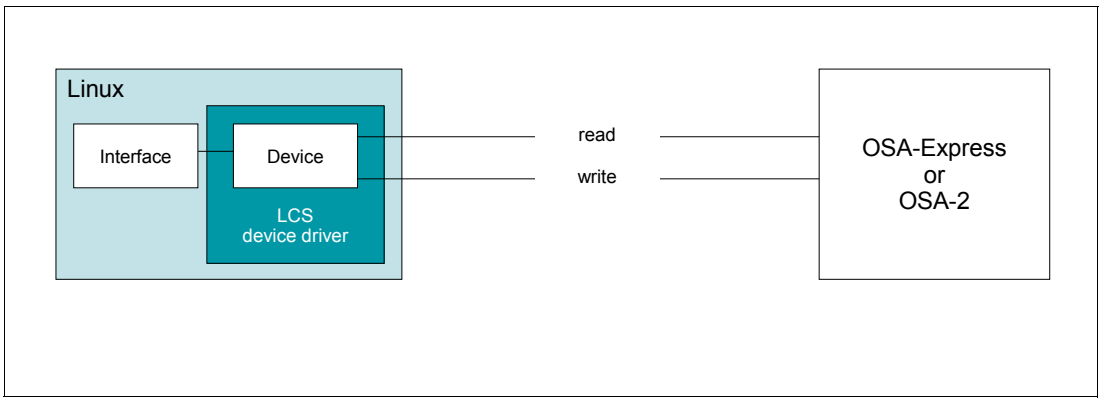


Figure 11-10 I/O subchannel interface

Figure 11-11 illustrates the much shorter I/O process when in QDIO mode compared with non-QDIO mode. I/O interrupts and I/O path-lenghts are minimized. This results in improved performance versus non-QDIO mode, reduction of System Assist Processor (SAP®) utilization, improved response time, and server cycle reduction.

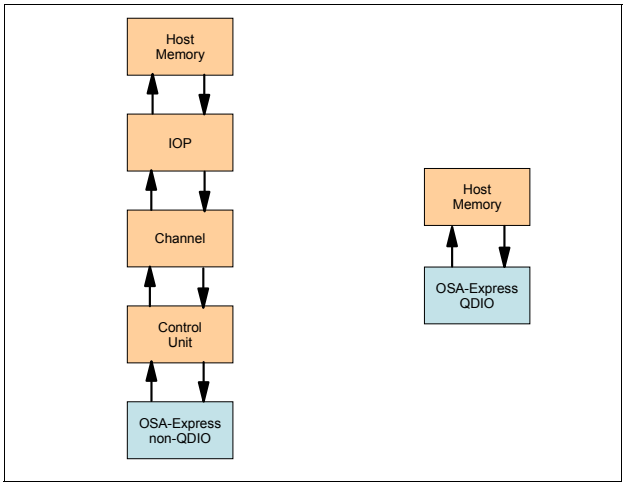


Figure 11-11 Non-QDIO versus QDIO data path

11.2.3 CTCMPC device driver

This driver was used (before VSWITCH) to provide LAN connectivity between Linux images and the rest of the network. With this configuration the z/VM TCP/IP stack was responsible for routing packets between Linux images and the outside network. This architecture, for performance reasons, is no longer used. You can still use this driver to perform point-to-point connection between Linux images or Linux systems under LPAR connected to other Linux images, or in LPAR installed on other System z machines.

This solution can provide network isolation and can be a very useful architecture for a high-availability solution based on heartbeat clustering methodologies.

The CTCMPC device driver requires two I/O subchannels for each interface, a read subchannel and a write subchannel. The device bus-IDs that correspond to the two subchannels must be configured for control unit type 3088.

Figure 11-12 shows the I/O subchannel interface.

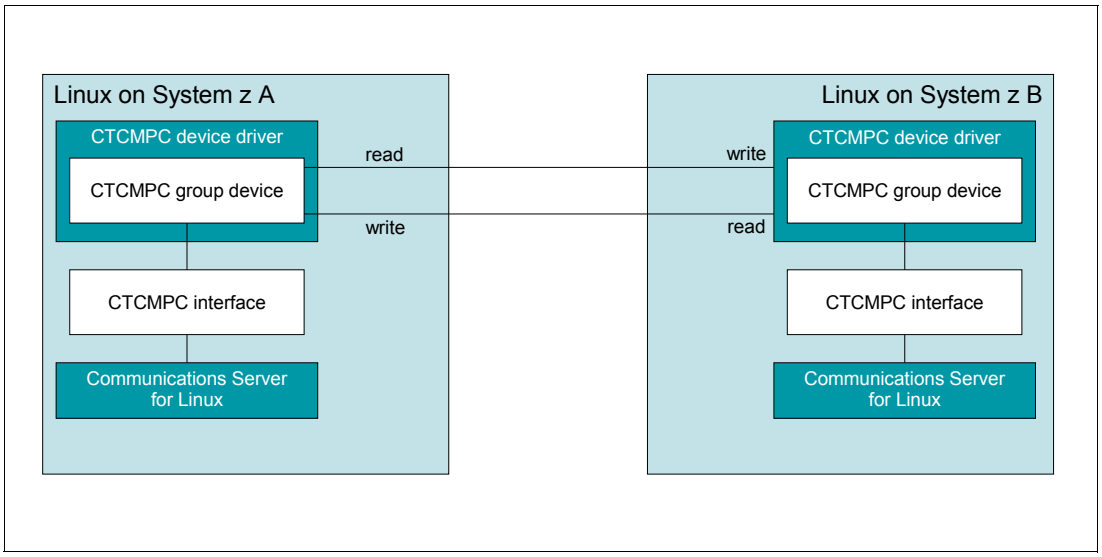


Figure 11-12 I/O subchannel interface

11.3 z/VM Virtual Switch link aggregation

Link aggregation support for the virtual switch was first introduced in z/VM starting with z/VM Release 5.3. The virtual switch can be configured to operate with aggregated links in concert with a switch that also supports the IEEE 802.3ad link aggregation specification. Aggregating links with a partner switch box allows multiple OSA-Express2 adapters to be deployed in transporting data between the switches. This configuration provides the benefits of increased bandwidth and near seamless recovery of a failed link within the aggregated group.

Figure 11-13 shows the architecture based on z/VM Virtual Switch Link Aggregation.

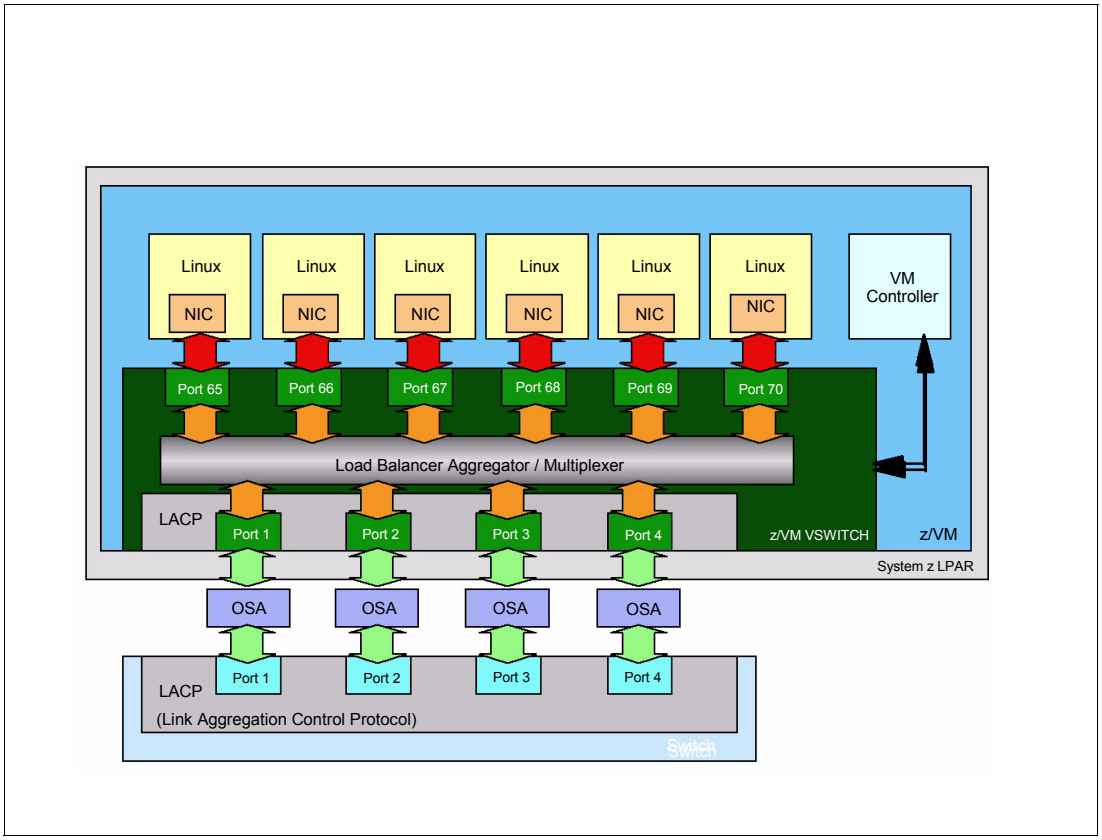


Figure 11-13 z/VM VSWITCH link aggregation architecture

Table 11-4 shows measurements made in an architecture based on link aggregation support with one or two OSA adapters configured. This shows the scaling up of throughput when more adapters are used. VSWITCH balances the workloads between the interfaces and Linux benefits from it.

Table 11-4 Link aggregation streams measurements

Test	Measure	One thread	Ten threads	Fifty threads
1 OSA MTU 1492	MBps	82.4	112	111.9
1 OSA MTU 1492	Total CPU	5.54	6.4	7.15
2 OSA MTU 1492	MBps	94.7	206.1	216.9
2 OSA MTU 1492	Total CPU	5.41	5.65	5.57
1 OSA MTU 8992	MBps	70.7	118	117.8
1 OSA MTU 8992	Total CPU	3.8	3.66	3.91
2 OSA MTU 8992	MBps	74.6	235.9	235.6
2 OSA MTU 8992	Total CPU	3.38	3.22	3.38

This results are clear in Figure 11-14. As you can see, throughput differences between one OSA and two OSAs increase when you increase the number of threads.

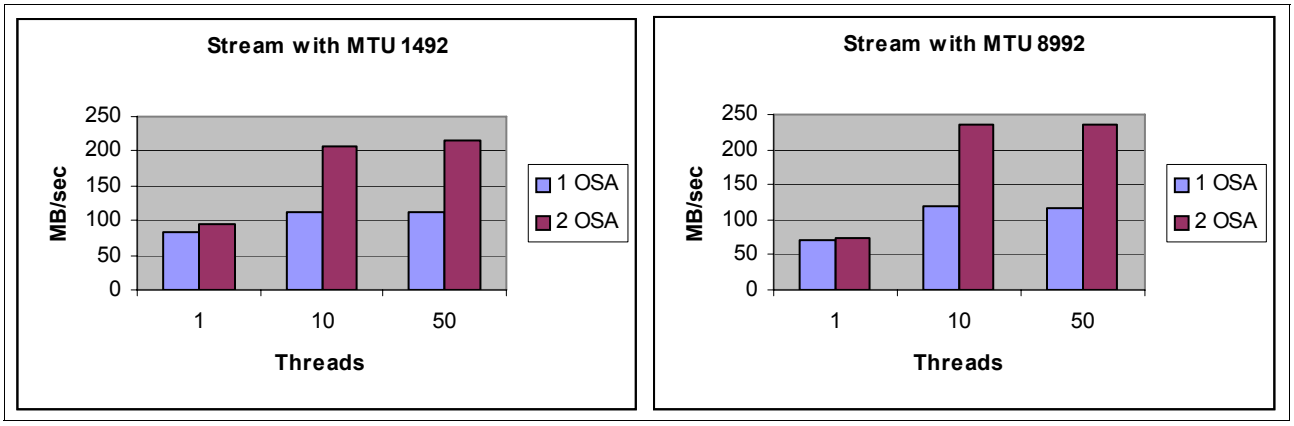


Figure 11-14 MBps comparison

At the same time, CPU cost is similar in the same situations. This means better results at no CPU cost. See Figure 11-15.

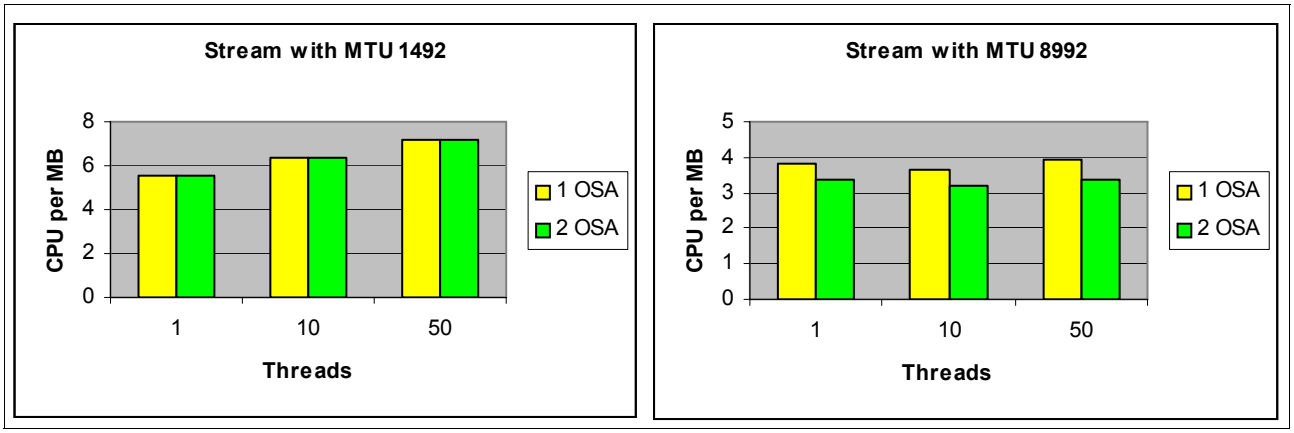


Figure 11-15 CPU per MB comparison

For more information about this performance test refer to:

http://www.vm.ibm.com/perf/reports/zvm/html/index.html

11.4 Hipersockets

Hipersockets is a technology that provides high-speed TCP/IP connectivity between servers within a System z. This technology eliminates the requirement for any physical cabling or external networking connection among these servers. It works as an internal Local Area Network, secure and protected by the entire external network. Hipersockets are very useful if you have a large amount of data exchange between partitions on the same hardware.

This architecture can leverage overall application performance, especially in solutions based on three-tiered architecture. Three-tier architecture is shown in Figure 11-16.

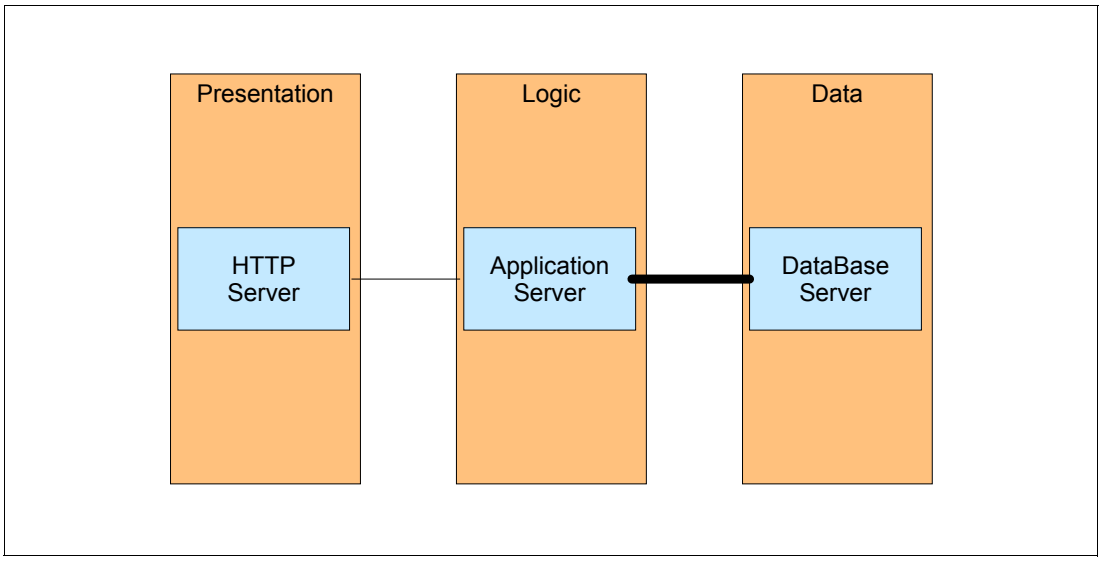


Figure 11-16 Three-tier architecture

This architecture is usually used in many eBusiness scenarios, for example, SAP. In this architecture the layers are interconnected with TCP/IP. This means that each server should be connected with a network. This architecture can introduce problem related to network when an high backend throughput is needed. This can be usual in many applications with an high use of databases. In fact, many applications may require a dedicated network for performance reasons between business logic and data. With Hipersockets technology, this connectivity is provided using an internal queued input/output (iQDIO) at memory speed to pass traffic among servers included in the same System z.

In Figure 11-17 we shows the difference between OSA cards and Hipersockets in a CRR benchmark test. The number of transactions we are performing (this is not a stress test) is not important, but evaluate the differences between the two technologies with the same benchmark characteristics.

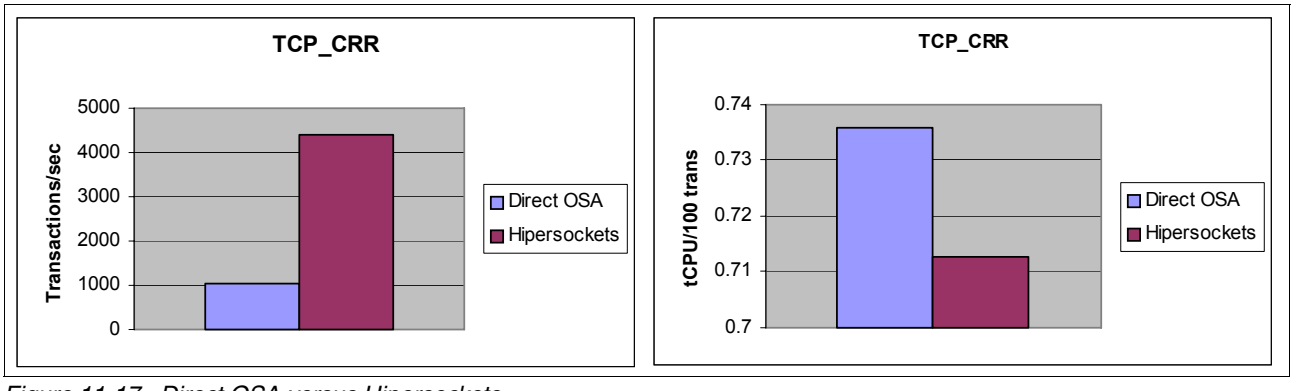


Figure 11-17 Direct OSA versus Hipersockets

11.5 High-availability costs

As discussed in many sections in this chapter, network availability is one of the most common problems related to servers. When you plan a Linux architecture on System z you should consider potential single points of failure. Networking high-availability scenarios may influence the overall performance of systems. If you are in the z/VM scenario, you can chose to have VSWITCH or Link aggregation support to perform an high-availability solution. It depends on your current System z machine, z/VM level, and network topology. If you are running Linux on LPAR you need to manage network failure with Linux software, for example, zebra. This software provides daemons useful to manage virtual IP addressing and dynamic routing. The following sections below show the costs of high-availability networking when you run your Linux systems in LPARs or as a guest of z/VM.

11.5.1 LPAR: static routing versus dynamic routing

In LPAR scenarios, you can architect a high-availability solution for adapters adopting VIPA. VIPA allows you to assign IP addresses that are not associated with a particular adapter. This address can be routed to one real interface. You can decide to create static routing to do that or use a dynamic routing protocol able to communicate with protocols in the network and update routing table.

Many distributions support quagga as the dynamic routing manager. Quagga use two different daemons:

- zebra

  It provides kernel routing updates, redistribution of routes between different routing protocols and interface lookups.

- ospfd

  Open shortest path first is a routing protocol classified as an interior gateway protocol. It distributes routing information between routers belonging to a single autonomous system.

Using a daemon for dynamic routing introduces workload to be managed. Daemons like quagga usually wake up periodically, sending packets to neighbors to maintain connectivity. This period is set on the hello-interval parameter and can influence overall performance.

Dynamic routing is important if you need a auto-managed high availability. If you use a static route, in case of loss of signals or some other problems related to the OSA interface card, you need to modify manually the routing configuration. The time between the problem being reported and the manual modification of the static route may influence the availability from the application and user point of view.

In this example we show the differences between the static route and the dynamic route in terms of throughput and CPU usage by the system. In this test we use connect request response reports.

Figure 11-18 shows the differences in terms of transaction per second. Static routing performs a little bit better the other two cases. In a dynamic routing configuration there are not significant differences if you change the hello-interval parameter. The number in parentheses in the graph legend indicates the hello-interval setting parameters.

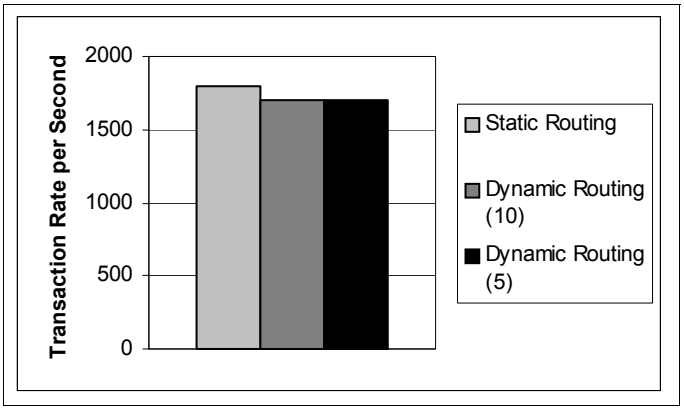


Figure 11-18 Transaction rate per second

Differences are evident in terms of CPU. The continuos polling of hello-interval influences the CPU used to manage the connections. Figure 11-19 shows these differences measured.

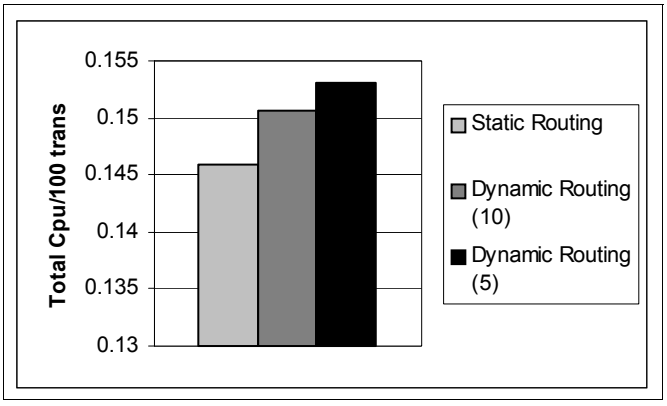


Figure 11-19 Total CPU/100 trans

11.5.2 z/VM: Linux router versus VSWITCH

Since z/VM 4.4.0, a feature called VSWITCH was introduced to provide a bridge to the network for Linux images. Before it there were solutions able to consolidate images under a z/VM Guest LAN routed to real network (LAN) with a Linux Router built in the z/VM Linux colony. This architecture was very useful, but introduced problems related to high availability. In fact, in order to provide high availability the Linux routers must be duplicated. Introducing two Linux images to provide connectivity also introduces an overhead to manage it. Using VSWITCH reduced this overhead because it provides itself with high availability between two different network adapters.

The virtual switch helps alleviate performance problems by moving data directly between the real network adapter and the target or originating guest data buffers.

Solutions based on Linux routers are still used when a Linux firewall is needed to control packets between the real network and the internal Linux network. If you are planning to use

the Linux router as a firewall remember that this can introduce overhead in terms of network throughput and total CPU usage.

In this scenario we tests different workloads:

- Request response (RR)

  The client sends 200 bytes to the server and the server responds with 1,000 bytes. This interaction lasts for 200 seconds.

- Connect request response (CRR)

  The client connects, sends 64 bytes to the server, the server responds with 8 K, and the client disconnects. The sequence is repeated for 200 seconds.

- Streaming (STREAM)

  The client sends 20 bytes to the server and the server responds with 20 MB. The sequence is repeated for 200 seconds.

The graph shown in Figure 11-20 compares results when running Linux TCP/IP communication through a Linux router and a virtual switch. For more information about these tests refer to:

http://www.ibm.com/perf/reports/zvm/html/vswitch.html

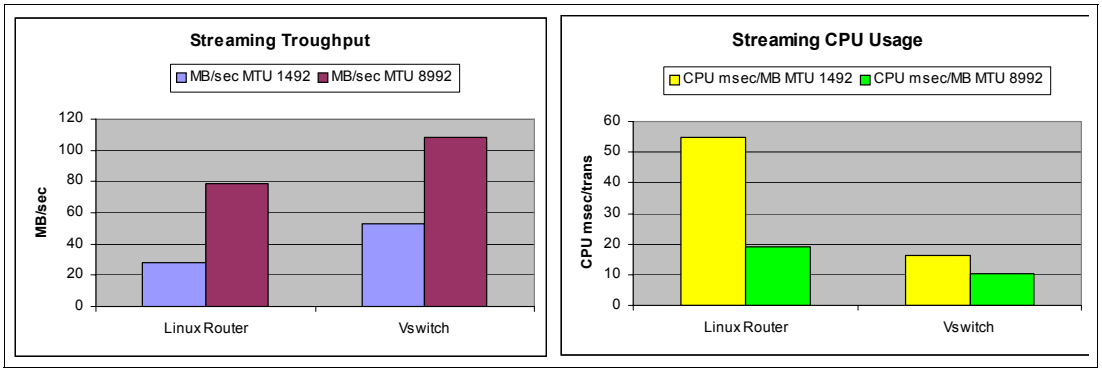


Figure 11-20 Streaming test result

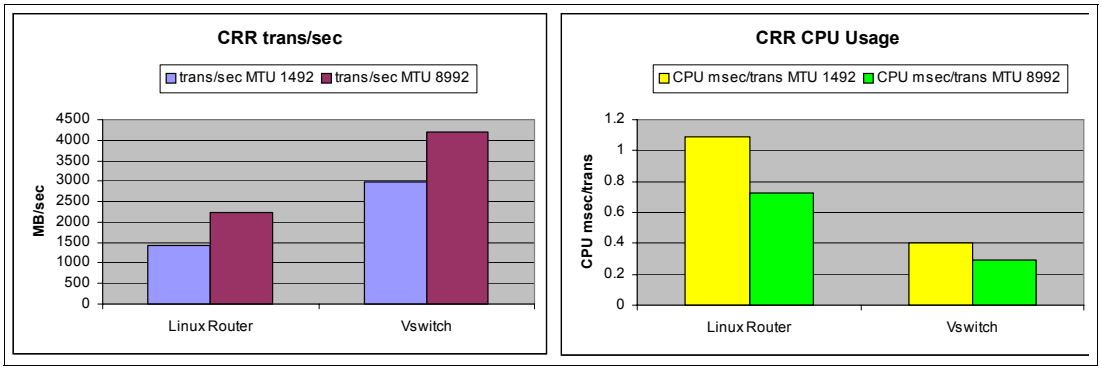


Figure 11-21 CRR test result

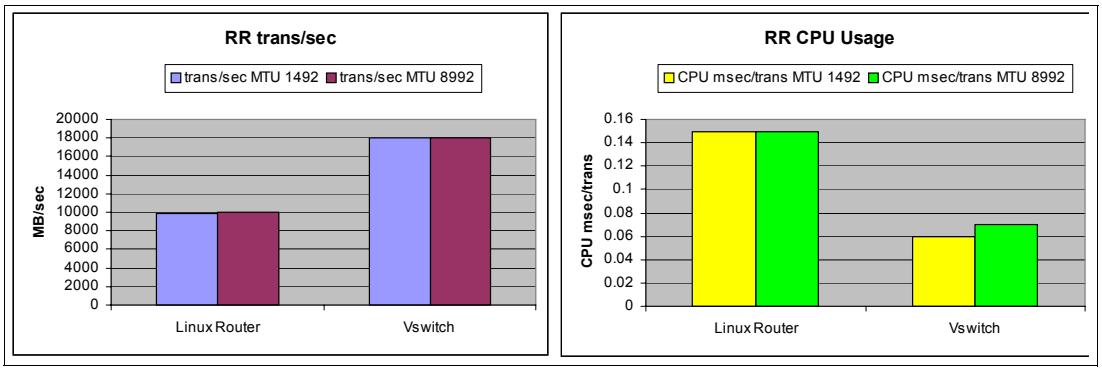


Figure 11-22 RR test results

11.6 Workloads and performance tools

As mentioned in many parts of this chapter, we made our tests using different workload scenarios. This scenarios reproduce real cases of a production network traffic. We use different tools to documents results. This tools are:

- netperf

  This is designed based on a client-server model. It is divided into two processes, one to manage the server (netserver), and a client to control benchmarks (netperf). For more details refer to:

  `http://www.netperf.org`

- iperf

  This tool is based on a client-server model and is useful for throughput calculation. For more information refer to

  `http://dast.nlanr.net/projects/Iperf/`

Workload used in our tests are:

- Request-response test (RR)

  This test describes data exchange like what usually happens in interactive environments, that is, long-running transport connections with small amounts of data being exchanged in a conversational way. For this workload, *networking performance* means transaction per wall clock second and CPU time consumed per transaction.

- Connect-request-response (CRR)

  This test describes data exchange that usually happens with an HTTP server. That is, at connection start, a small amount of data flows in, a moderate amount flows out, and the connection then ends. For this workload, we are still interested in transactions per wall clock second and CPU time consumed per transactions (or multiple).

- Streaming get (STRG)

  In this test, the connection is persistent, just as in RR. However, the data sent in one direction is very small (just a few byes), and the data sent in the other direction is very large. This scenario illustrates the best-case rate at which the computing system can pump data across a communication link. In this environment what interests us are megabytes per wall clock second and CPU time used per megabyte send.

A

WebSphere performance benchmark sample workload

In this IBM Redbooks publication, we simulated measurable workloads for each example using the Trade Performance Benchmark Sample for WebSphere Application Server Version 6.1.0.11(Trade 6) as the sample application. This appendix describes the sample application.

© Copyright IBM Corp. 2003, 2008. All rights reserved.

IBM Trade Performance Benchmark sample

The IBM Trade Performance Benchmark sample, also known as the Trade 6 application, is a sample WebSphere end-to-end benchmark and performance sample application. This Trade benchmark has been designed and developed to cover WebSphere's programming model. This application provides a real-world workload, driving WebSphere's implementation of J2EE 1.4 and Web services, including key WebSphere performance components and features.

The Trade 6 application simulates a stock trading application that allows you to buy and sell stock, to check your portfolio, register as a new user, and so on. In this IBM Redbooks publication, we use Trade 6 to generate workloads that are analyzed in terms of their impact on system performance. The IBM Trade Performance Benchmark sample for WebSphere Application Server is available to download for free at:

`https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=trade6`

The three main Trade 6 components are listed below and shown in Figure A-1:

- The IBM HTTP server

  The HTTP server accepts client requests and delivers static content (HTML pages, images, and style sheets). Dynamic requests are forwarded to the WebSphere Application Server through a server plug-in.

- The IBM WebSphere Application Server

  The WebSphere Application Server creates dynamic content using JavaServer™ Pages™ (JSPs) and Java Servlets. Pages are generated from data extracted from a DB2 database. All Trade-versions are WebSphere-version dependent, so Trade 6 only works with WebSphere Application Server V6.

- The DB2 database

  The DB2 database contains relational tables regarding simulated customer accounts and stock transactions. We used DB2 LUW v8.2.

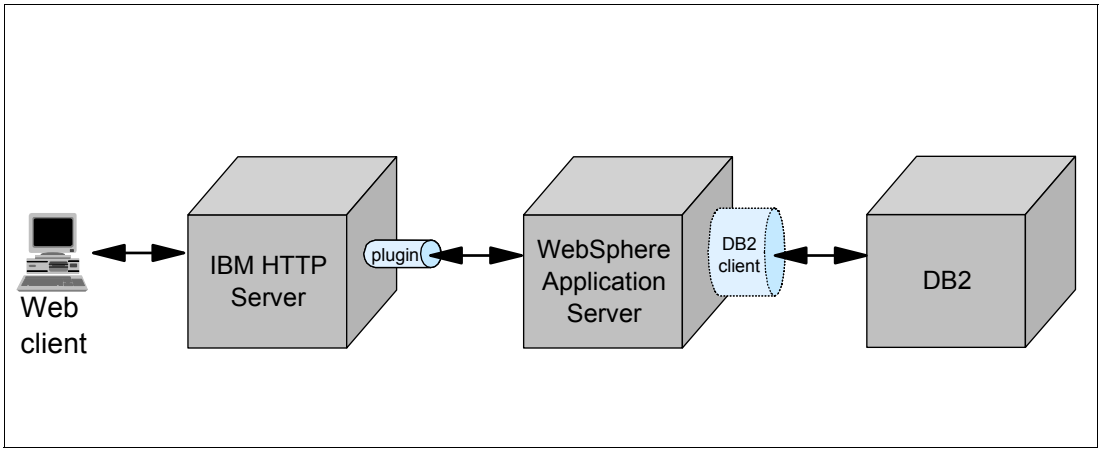


Figure A-1 Components of a typical Trade 6 deployment

Trade 6 Deployment options

Several options are available when deploying Trade 6 on Linux for System z:

- All components can run in a single Linux guest. This is referred to as a single-tier deployment.
- Each component can run in a dedicated Linux guest. We refer to this as a three-tier deployment.

We chose a three-tier deployment for this book, where we run the IBM HTTP server, the WebSphere Application Server, and DB2 in their own Linux guests. Our deployment choice is depicted in Figure A-2.

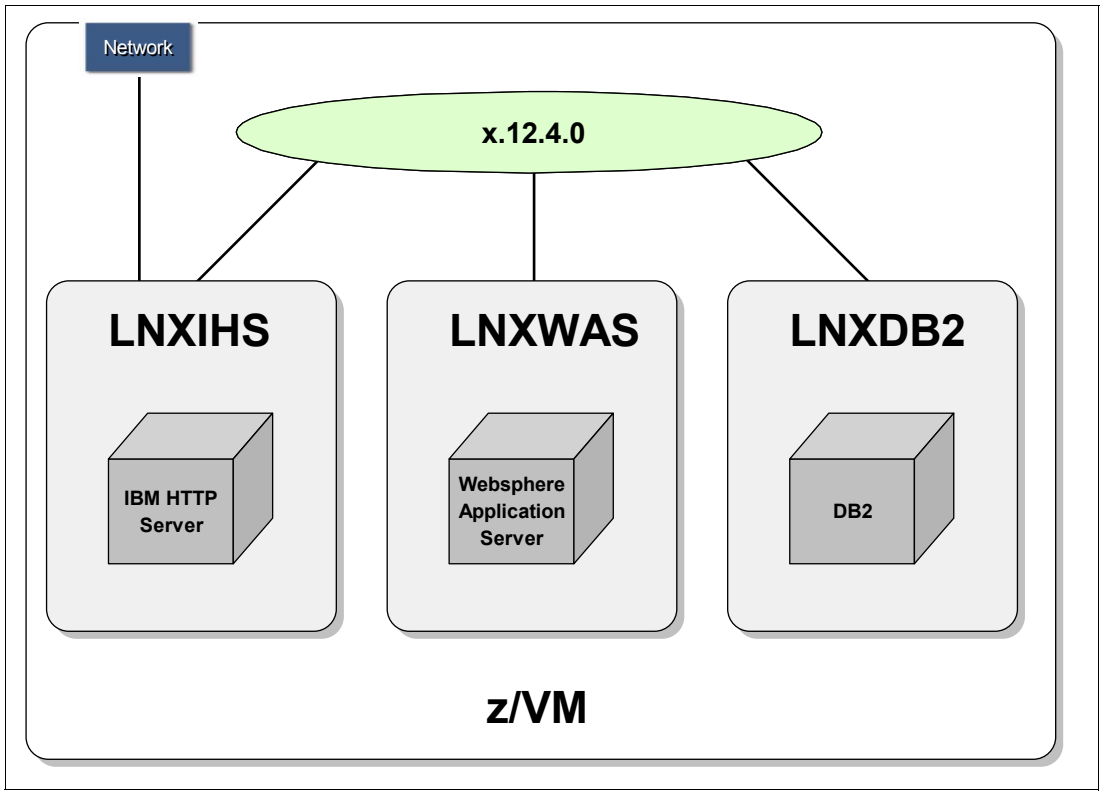


Figure A-2 Three-tier deployment

Using a three-tier deployment enabled us to adjust the virtual machine size of each Linux guest more accurately, based on the task that particular guest performs. In addition, when measuring utilization, the three-tier deployment option allows us to attribute specific resource usage to a specific task.

B

WebSphere Studio Workload Simulator

This appendix describes our use of the WebSphere Studio Workload Simulator as a workload generator. Topics discussed include an overview of the WebSphere Studio Workload Simulator and the sample workload generation script we used in conjunction with the Trade 6 WebSphere application (see Appendix A, “WebSphere performance benchmark sample workload” on page 191, for more information about Trade 6).

© Copyright IBM Corp. 2003, 2008. All rights reserved.

WebSphere Studio Workload Simulator overview

The WebSphere Studio Workload Simulator for z/OS and OS/390 is an application that allows you to create multiple *virtual* or *simulated* users in order to test load and performance.

The Workload Simulator consists of two components: a controller and an engine. For high scalability, Workload Simulator's Engine, which generates the load used during load-testing, is installed on a System z server. The load generated by the engine can be used to test any Web-serving environment (that is, the environment to be tested is not limited to z/OS). In our case, we tested against a WebSphere Application Server that was running on a Linux on System z guest. Workload Simulator supports multiple engines.

Sample workload generation script

This section outlines the script that we used to generate a workload against the WebSphere Application Server. See Appendix E, “Additional material” on page 211, for information about how to download the script that we used. In addition, you may want to download the MultiClients.conf configuration file. You also need to create an engine definition.

After you download the script, which is shown in Example B-1, you modify the string host name to reference your own host name.

Example: B-1 Sample Trade 6 workload generation script

```
//
/*trade6 version 1.2*/
init_section
{
   string hostname = "x.12.4.xxx:9080";
   //string hostname = "x.12.4.xxx";
   int botClient = 0;
   int topClient = 3999;
   shared int curClient = botClient - 1;
}
int html_percent = 10;
int gif_percent = 0;
//repopulate and reconfigure trade on the website to reflect the following numbers
int num_u = 4000;
int num_s = 2000;
int num_stock, i;
string name, uid, add, email, holdid, stocks;
bool loop = true;
int sell_deficit = 0;
HttpResponse r;
start_transaction("login");
   startpage(4);
   thinktime(1000);
      //uid = URLEncode("uid:"+random(0,num_u - 1));
      //TODO: Make this random but better than above - must be random across all
clients
      int clientnum;
      enter_critical_section();
      curClient = curClient + 1;
      if (curClient > topClient)
```

```
      {
         curClient = botClient;
      }

      clientnum = curClient;
      leave_critical_section();
      uid = URLEncode("uid:" + clientnum);
      postpage(""+hostname+"","/trade/app",1,close,0,start,"",
      "uid=" +uid+ "&passwd=xxx&action=login",
      "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
      "Referer: http://"+hostname+"/trade/app",
      "Accept-Language: en-us",
      "Content-Type: application/x-www-form-urlencoded",
      "Accept-Encoding: gzip, deflate",
      "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
      "Host: "+hostname+"",
      "Connection: Keep-Alive");

   endpage;
end_transaction("login");
while (loop)
{
   distribute
   {
      weight 20:
         start_transaction("home");
            startpage(5);
            thinktime(1000);
               getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=home",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
         end_transaction("home");

      weight 4:
         start_transaction("update_account");
            // Performan an account, followed by an account update
            startpage(6);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=account",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Content-Type: application/x-www-form-urlencoded",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");
```

```
            endpage;
            // update the account to some random data
            startpage(7);
            thinktime(5000);
               getpage(""+hostname+"","/trade/app",1,close,0,start,"?userID="
+uid+ "&fullname=" +"rnd"+current_client()+now()+
"&password=xxx&address=rndAddress&cpassword=xxx&creditcard=rndCC&email=rndEmail&ac
tion=update_profile",
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

            endpage;
         end_transaction("update_account");

      weight 10:
         start_transaction("account");
            startpage(6);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=account",
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

            endpage;
         end_transaction("account");

      weight 4:
         start_transaction("sell");
            startpage(8);
            thinktime(1000);
               r =
getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=portfolio",
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

            endpage;
            holdid = r.extractVariable("holdingID=","\"");
            if (StrLength(holdid) > 0)
            {
```

```
               startpage(9);
               thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=sell&holdingID="
+holdid,
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

               endpage;
            }
            else
            {
               // If the user has no holdings, switch to a buy and increment
sell_deficit counter
               sell_deficit = sell_deficit + 1;
               stocks="s:"+random(0, num_s-1);
               startpage(10);
               thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=quotes&symbols="
+stocks,
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

               endpage;
               startpage(11);
               thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=buy&symbol=" +stocks+
"&quantity=" +random(1,
                  200),
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");
```

```
                endpage;

             }

          end_transaction("sell");

       weight 12:
          start_transaction("portfolio");
             startpage(8);
             thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=portfolio",
                "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                "Referer: http://"+hostname+"/trade/app",
                "Accept-Language: en-us",
                "Accept-Encoding: gzip, deflate",
                "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

             endpage;
          end_transaction("portfolio");

       weight 4:
          stocks="s:"+random(0, num_s-1);
          start_transaction("buy");
             if ( sell_deficit <= 0 )
             {
                startpage(10);
                thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=quotes&symbols="
+stocks,
                   "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                   "Referer: http://"+hostname+"/trade/app",
                   "Accept-Language: en-us",
                   "Accept-Encoding: gzip, deflate",
                   "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                   "Host: "+hostname+"",
                   "Connection: Keep-Alive");

                endpage;
                startpage(11);
                thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=buy&symbol=" +stocks+
"&quantity=" +random(1,
                   200),
                   "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                   "Referer: http://"+hostname+"/trade/app",
                   "Accept-Language: en-us",
                   "Accept-Encoding: gzip, deflate",
```

```
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

               endpage;
            }
            else
            {
               // If there is a sell deficit, perform a sell instead to keep
buys/sells even
               startpage(8);
               thinktime(1000);
                  r =
getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=portfolio",
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

               endpage;
               holdid = r.extractVariable("holdingID=","\"");
               if (StrLength(holdid) > 0)
               {
                  sell_deficit = sell_deficit - 1;
                  startpage(9);
                  thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=sell&holdingID="
+holdid,
                     "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                     "Referer: http://"+hostname+"/trade/app",
                     "Accept-Language: en-us",
                     "Accept-Encoding: gzip, deflate",
                     "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                     "Host: "+hostname+"",
                     "Connection: Keep-Alive");

                  endpage;
               }


            }

         end_transaction("buy");

      weight 40:
         start_transaction("quotes");
```

```
            stocks="s:"+random(0, num_s-1);
            startpage(10);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=quotes&symbols="
+stocks,
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
         end_transaction("quotes");

      weight 2:
         start_transaction("register");
            //log the current user out and register a new user
            startpage(12);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=logout",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
            clear_cookie_cache();
            // load the registration page
            startpage(13);
            thinktime(1000);
               getpage(""+hostname+"","/trade/register.jsp",1,close,0,start,"",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
            // register a new user and login
            name = URLEncode("first:"+random(0,999)+" last:"+random(0,4999));
            add = URLEncode(random(1, 5000) + " mystreet");
            uid = URLEncode("ru:"+current_client()+":"+now());
            email = URLEncode(uid+"@"+random(0,100)+".com");
            startpage(14);
            thinktime(1000);
```

```
               getpage(""+hostname+"","/trade/app",1,close,0,start,"?Full+Name="
+name+ "&snail+mail=" +add+ "&email=" +email+ "&user+id=" +uid+
"&passwd=yyy&confirm+passwd=yyy&money=1000000&Credit+Card+Number=123-fake-ccnum-45
6&action=register",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
         end_transaction("register");

      weight 4:
         start_transaction("logoff");
            startpage(12);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=logout",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
            clear_cookie_cache();
         end_transaction("logoff");
         loop = false;

   }
}

stop;
```

For more details about this product, see:

http://www-306.ibm.com/software/awdtools/studioworkloadsimulator/about/

There is a cost associated with the use of this product. Contact your IBM representative for more information.

C

Mstone workload generator

This appendix describes the Mstone workload generator. Topics discussed include:

- Mstone overview
- Operation of the Mstone workload

© Copyright IBM Corp. 2003, 2008. All rights reserved.

Mstone overview

Mstone is a performance measurement tool available from the Mozilla project:

http://sourceforge.net/projects/mstone

Originally known as Mailstone, Mstone can be used for capacity planning and testing network mail servers. The Mailstone user guide can be found at:

http://mstone/sourceforge.net//

Using Mstone, workloads can be generated on mail servers from multiple client machines. Test scenarios that employ multiple mail protocols and multiple simulated mail clients can be used to observe server response under heavy load. Although originally intended to test mail server performance, Mstone has features that make it suitable for overall system performance analysis. The Mstone configuration used in this book is shown in Figure C-1.

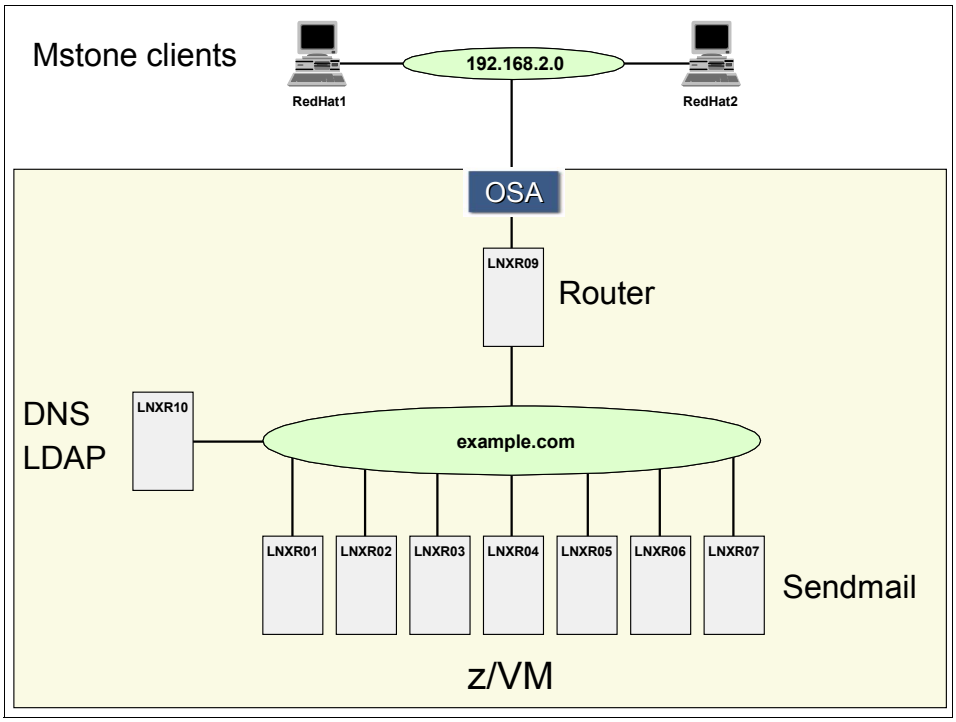


Figure C-1 Mstone workload configuration

D

Emergency scanning and below 2 GB line constraint

In this appendix we discuss emergency scanning, which may arise when constraint is present with a virtual machine, below the 2 GB line.

In this appendix, the following topics are discussed:

- What is emergency scanning
- Below 2 GB line constraint and some of the scenarios in which is has been seen
- The fixes within VM Releases that have removed much of the below 2 GB line constraint
- Scenarios where you may still experience emergency scanning

© Copyright IBM Corp. 2003, 2008. All rights reserved.

Emergency scanning

Emergency scanning is an indicator that the system is critically short of storage for some reason. When short of storage pages, the system begins a three-level scanning process: scan 1, scan 2, and then the most serious, emergency scanning. The system is being reduced to extreme measures while hunting for main storage pages, and the situation is so severe that it will steal them from any user. Ultimately, whether they are inactive, in queue, or active, it eventually even steals pages from shared segments and the CP system process. In short, the system is in a critical mess, storage wise. This also covers stealing pages from the monitor and MONDCSS, two critical performance-related entities, and in this event, performance monitoring and reporting may be compromised or interrupted.

Below 2 GB line constraint and scenarios

VM has enjoyed a long and distinguished history since the earliest versions were released over 40 years ago. In the world of Virtualization today, it is the most functional, flexible, customizable operating system to be found on any platform. The latest Version 5 levels enjoy reliability and stability levels other platforms can only remotely aspire to.

A major step forward in this evolution took place in v3.1.0, which introduced the initial 64-bit support. The next major milestone in this regard was v5.2.0, which added a number of major enhancements to 64-bit support.

Prior to this, some parts of VM 64-bit releases were still operating in 31-bit mode, which implies an upper addressability limit of 2 GB. This could cause certain runtime issues when running Linux guests, for example, when they were performing I/O. Linux has the unfortunate tendency to grab and hold on to all of the memory allocated to it, which sometimes leads to a large part of it being misused for internal I/O buffer and cache. Linux guests, where a virtual storage of up to or greater than 2 GB is defined, are now common on many systems. In order to perform real I/O in this particular scenario prior to VM v5.2.0, the system needed to use areas below the requesting virtual machine 2 GB line, and, as these were already in use by the virtual machine itself, the system has to resort to emergency scanning to find the storage pages that it requires to complete the I/O task.

Figure D-1 on page 209 shows an example of a system in emergency scanning mode. This system was running z/VM v4.4.0. Some experiments were carried out by using a different value for test idle time, which is preset at 300 milliseconds within VM. Note that this is not a standard release function, and it was used here strictly to understand the nature of the problems the system was encountering, and then to reduce their impact. For various reasons, the system owner was unwilling to move forward to z/VM v5, which at the time would have removed many of the reasons for ESCN being observed. But could we still tune the system as it stood? This testing was carried out at the end of February on the chart, and you can see the dramatic effects. With test idle reduced to 10 ms, the queue drop became much more effective, and the emergency scanning stopped. You can see the effects on the non trivial transaction time (NT-T), which is often used as a function of response time for many systems. Here we are using it for a different purpose, to show that queue drop is taking place, resulting in shorter transactions. Before any change was made to the system, NT-T was 30 seconds on average. After the change, this goes down to typically 1 second or less. The throughput of the system was much improved. However, some of this improvement is offset by the removal of the known beneficial effects of having test idle set at 300 ms. This enables many resource cycles to be saved by not dropping a virtual machine from the queue that becomes active again in the test idle period. The system then does not have to go through the process of bringing a virtual machine through all the various stages necessary to enable it to dispatch.

Figure D-1 shows attempts made in late March to recover some of these benefits, while still getting the Linux guest to queue drop, but the emergency scanning soon starts to reappear.

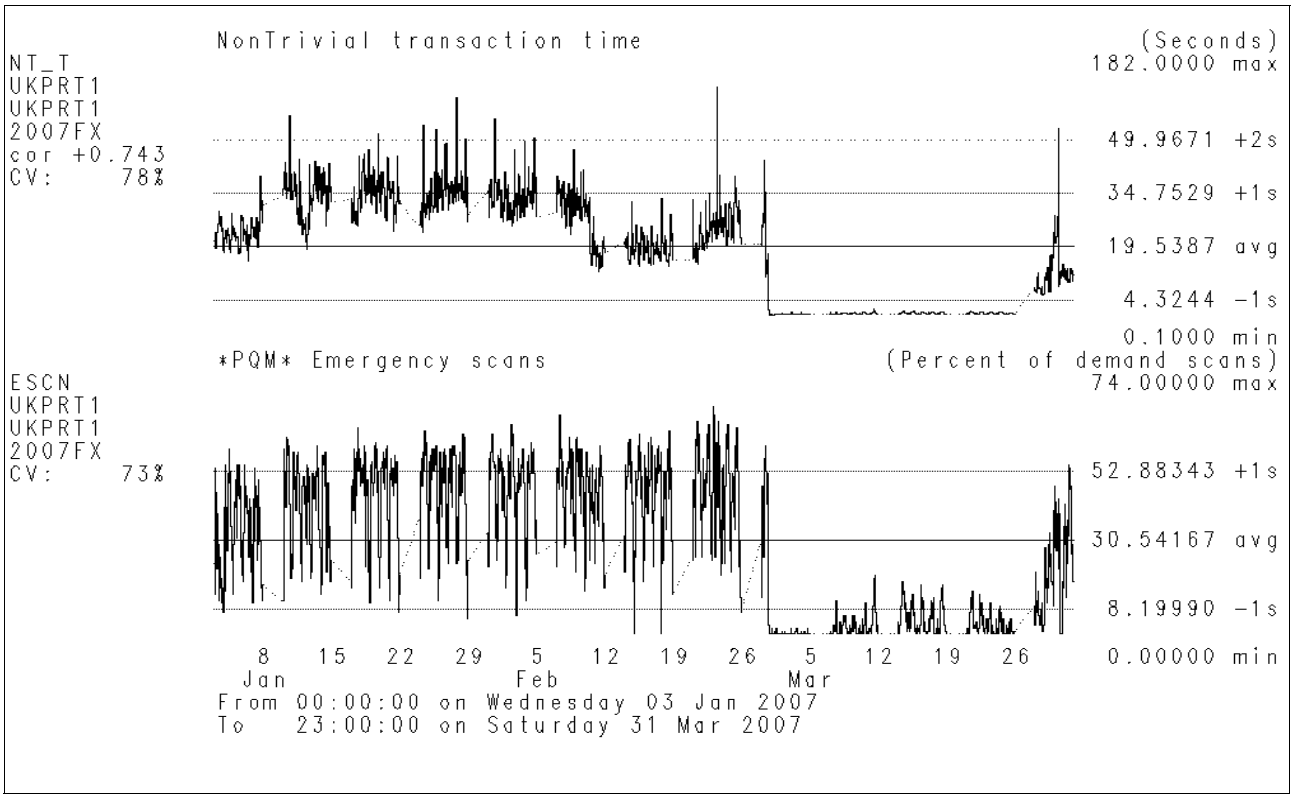


Figure D-1 Example of System in Emergency Scanning mode

Fixes within VM releases that have removed much of the below 2 GB line constraint

The z/VM v5.2.0 release made considerable changes that allowed large real storage sizes to be more fully exploited for the first time. The key to these changes was the introduction of System Execution Space (SXS). CP runs in an address space of its own, which can be up to 2 GB in size. Before this release, SXS was identity mapped, and all logical addresses exactly corresponded to real addresses. When needing to reference a guest page, CP now uses mapping to a logical page in SXS. Most of the CP code can continue to run using 31-bit addressability, but most importantly, this removes the requirement to move pages to real frames below the 2 GB line, so CP code and also most CP data structures can now be positioned above the 2 GB line. At this release, the maximum real storage size for VM was 128 GB.

z/VM v5.3.0 introduced more important enhancements to CP storage management. Page management blocks can now be positioned above the 2 GB line. Contiguous frame management has been further refined, and fast available list searching was introduced in this release. The maximum storage size for VM is now 256 GB. For more discussion and related performance test results see:

http://www.ibm.com/perf/reports/zvm/html/530stor.html

Scenarios where you may still experience emergency scanning

In the test runs carried out in the writing of this book, we observed some emergency scanning. In Figure D-2, a Linux guest had just been heavily committed in terms of memory within a 3 GB virtual machine, by defining a very large heap size for running the WebSphere Application Server. Some steps that you can take to generally tune your system, and help avoid ESCN where it does still arise, can be found at:

http://www.vm.com/perf/tips/2gstorag.html

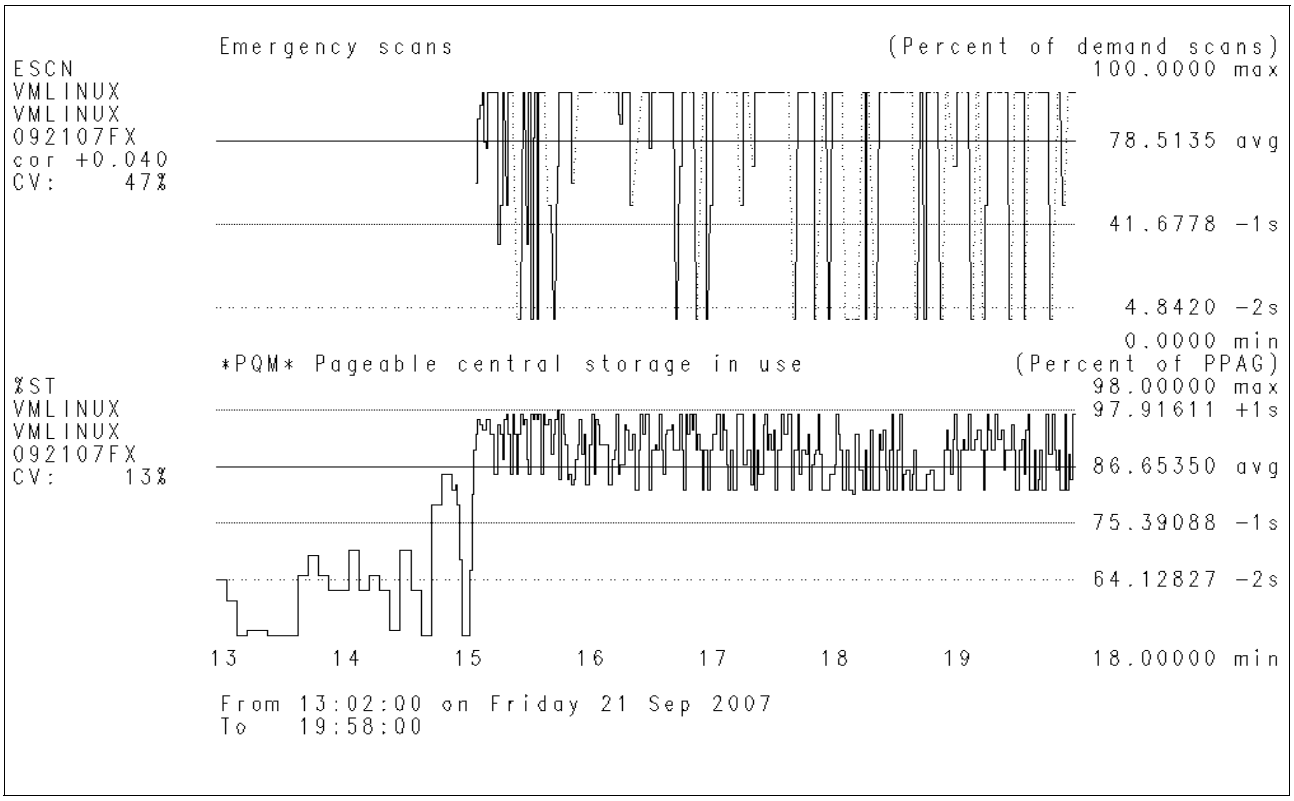


Figure D-2 Example emergency scanning

E

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG246926

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG24-6926.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
SG24692601.zip	Zipped samples

System requirements for downloading the Web material

The following system configuration is recommended to just run the code. You may need more to deploy and run the Trade 6 application:

Hard disk space: 3 MB minimum
Operating system: Windows or Linux

© Copyright IBM Corp. 2003, 2008. All rights reserved.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 214. Note that some of the documents referenced here may be available in softcopy only.

- *Linux for IBM System z9 and IBM zSeries*, SG24-6694
- *Linux Performance and Tuning Guidelines*, REDP-4285
- *z/VM and Linux on IBM System z: The Virtualization Cookbook for SLES9*, SG24-6695
- *Linux on IBM eServer zSeries and S/390: Performance Toolkit for VM*, SG24-6059

Other publications

These publications are also relevant as further information sources:

- *z/VM Performance Toolkit Guide,* SC24-6156
- *z/VM Performance,* SC24-6109
- *z/VM Getting Started with Linux on System z,* SC24-6096
- *IBM Tivoli OMEGAMON XE on z/VM and Linux, User’s Guide,* SC32-9489
- *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8289-03

Online resources

These Web sites are also relevant as further information sources:

- Introduction to the New Mainframe: z/VM Basics:
  http://www.redbooks.ibm.com/redpieces/abstracts/sg247316.html?Open
- Linux on System z library:
  http://www-03.ibm.com/systems/z/os/linux/library/
- IBM: VM performance documents:
  http://www.ibm.com/servers/eserver/zseries/zvm/perf/docs/
- IBM: VM performance tips:
  http://www.vm.ibm.com/perf/tips/
- IBM: Linux on System z - performance hints and tips
  http://www.ibm.com/developerworks/linux/linux390/perf/index.html
- IBM Linux on System z - development home page
  http://www.ibm.com/developerworks/linux/linux390/index.html

© Copyright IBM Corp. 2003, 2008. All rights reserved.

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

© Copyright IBM Corp. 2003, 2008. All rights reserved.

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Linux on IBM System z: Performance Measurement and Tuning

Understanding Linux performance on System z

z/VM performance concepts

Tuning z/VM Linux guests

In this IBM Redbooks publication we examine performance measurement and tuning for running Linux as a z/VM guest on an IBM System z z9 machine with a focus on the SLES 10 2.6 kernel and z/VM 5.3.

This book is intended for system administrators and IT architects responsible for deploying Linux servers running under z/VM. We examine performance concepts and identify tuning parameters that influence system performance. Using examples, we investigate performance and tuning topics for the memory, processor, DASD, and networking subsystems. Performance is analyzed at both the z/VM and Linux level. We provide tuning recommendations and guidance intended to maximize investment in Linux for System z.

The system used in the writing of this book is an IBM System z running z/VM Version 5.3 in an LPAR. The Linux distribution used in this book is Novell SUSE Linux Enterprise Server 10 (based on a Linux 2.6 kernel).

The intent of this book is to provide guidance on measuring and optimizing performance using an existing System z configuration. The examples demonstrate how to make effective use of your System z investment. The workloads used are chosen to exercise a specific subsystem.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6926-01 ISBN 0738485586